Modeling and Simulation Methods of Neuronal Populations and Neuronal Networks

by

Ningyuan Wang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Applied and Interdisciplinary Mathematics)
in The University of Michigan
2021

Doctoral Committee:

Professor Victoria Booth, Co-Chair
Professor Daniel B Forger, Co-Chair
Professor Dennis S. Bernstein
Professor Robert Krasny
Assistant Professor Adam Stinchcombe

Ningyuan Wang
nywang@umich.edu
ORCID iD: 0000-0002-3143-0180

To the deterministic or random laws of the universe that give rise to this work, even though they could not care less.

ACKNOWLEDGEMENTS

The motivation that leads me to the research was a self-contradictory one: using scientific evidence to prove that human's motive does not have any impact on the future of the universe, as the law of the universe, be it deterministic or random, does not leave a place for subjective inputs. Consequently, the work presented here is ultimately just the results of the law. As a social construct, we are accustomed to contributing our work as a result of the help of a certain entity. The law of the universe would seem to qualify as the most appropriate choice automatically.

However, such an abstraction would only paint a despairing and meaningless picture that one would like to escape from. In a more constrained fashion, let me constrain my gratitude to people only. I would first be very grateful for Yuanliang Zhao, who has been my close friend since secondary school, and we exchanged a lot of heated discussions on a lot of philosophical questions. I would almost certainly be pursuing a very different career had those discussions not took place.

The above only discussed the motive part of the work. As we know, there exists a multitude of examples where highly motivated people generate only pseudoscience or even less than that. The guidance and suggestions from the scientific community convert the motivation into a somewhat true result that is presented here. I would like to thank Prof. Daniel B. Forger, Prof. Adam Stinchcombe, and Prof. Victoria Booth for their guidance and encouragement throughout my Ph.D. years. And I thank Prof. Dennis S. Bernstein and Prof. Robert Krasny for their accurate and insightful suggestions and critiques in their field

of expertise. I am very grateful to all of them for serving on my dissertation committee. I also owe my thanks to James Hazelden, a talented and hardworking fellow who did a lot of the heavy-lifting parts in the Neuronal Network model and is a great pleasure to work with.

Finally, I would like to thank my parents, Liwei Li, and Huiping Wang. Their caring yet relaxed parenting shapes my personality more than anyone else.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This thesis presents numerical methods and modeling related to simulating neurons. Two approaches to the simulation are taken: a population density approach and a neuronal network approach.

The first two chapters present the results from the population density approach and its applications. The population density approach assumes that each neuron can be identified by its states (e.g., membrane potential, conductance of ion channels). Additionally, it assumes the population is large such that it can be approximated by a continuous population density distribution in the state space. By updating this population density, we can learn the macroscopic behavior of the population, such as the average firing rate and average membrane potential. The Population density approach avoids the need to simulate every single neuron when the population is large. While many previous population-density methods, such as the mean-field method, make further simplifications to the models, we developed the Asymmetric Particle Population Density (APPD) method to simulate the population density directly without the need to simplify the dynamics of the model. This enables us to simulate the macroscopic properties of coupled neuronal populations as accurately as a direct simulation.

The APPD method tracks multiple asymmetric Gaussians as they advance in time due to a convection-diffusion equation, and our main theoretical innovation is deriving this update algorithm by tracking a level set. Tracking a single Gaussian is also applicable to the Bayesian filtering for continuous-discrete systems. By adding a measurement-update step, we reformulated our tracking method as the Level Set Kalman Filter(LSKF) method and

find that it offers greater accuracy than state-of-the-art methods.

Chapter IV presents the methods for direct simulation of a neuronal network. For this approach, the aim is to build a high-performance and expandable framework that can be used to simulate various neuronal networks. The implementation is done on GPUs using CUDA, and this framework enables simulation for millions of neurons on a high-performance desktop computer. Additionally, real-time visualization of neuron activities is implemented. Pairing with the simulation framework, a detailed mouse cortex model with experiment-determined morphology using the CUBIC-Atlas, and neuron connectome information from Allen's brain atlas is generated.

CHAPTER I

Overview

The thesis consists of three main chapters. Each chapter presents a method or model that is relatively self-contained. Their goal is to develop methods for the simulation of neurons, but they can be helpful for a wide range of applications. The chapters are formulated in a way with their target audience in mind.

I.1: Overview of Chapter II

This chapter presents the level set Kalman filter (LSKF). This chapter is based on a submitted manuscript [WF21b], accompanied by additional discussions. We present a novel extension of Kalman filtering for continuous-discrete systems with nonlinear state-space models.

The LSKF assumes the probability distribution can be approximated as a Gaussian and updates the Gaussian distribution through a *time-update* step and a *measurement-update* step. This type of filtering method is well-established [KB61], with some rather recent developments [JU04, AHH10, KK17]. The LSKF improves the time-update step when compared to existing methods, such as the continuous-discrete cubature Kalman filter (CD-CKF) [AHH10] by reformulating the underlying Fokker-Planck equation as an ordinary differential equation for the Gaussian, thereby avoiding the need for expansion in time. With a carefully picked measurement-update method, numerical experiments show that the LSKF has a consistent performance improvement over CD-CKF for a range of parameters while simplifying the implementation, as no user-defined timestep subdivision between measure-

ments is required, and the spatial derivatives of the drift function are not explicitly needed.

I.2: Overview of Chapter III

This chapter presents the Asymmetric particle population density (APPD) method. Part of this work is drawn from [WF21a], while some of the applications are presented here for the first time.

The APPD method simulates a population of individual units. A common strategy to simulate such a system is to take a population density approach, where we take the macroscopic limit of the number of units and consider its population density function instead of individual units. In many of the previous applications of this approach in the context of neuronal populations [Win67, JR95, San96, NT00, Kur03], they also require further simplifications of the model. Our method uses Gaussians as particles to accurately simulate both the dynamics and noise of the units. While the idea of using a deforming Gaussian to solve a convection-diffusion equation is not new [Ros96, Ros05, BBH21], our method is the first that is applied to 4-dimensional state space, whereas most previous methods are restricted to 2 dimensions. Moreover, the APPD method is well-suited for a parallel implementation. We developed a *C++* library that implemented this method in parallel, which can be adapted to a range of models with relative ease. The library is available on `github` under the MIT license. To show the effectiveness and robustness of the APPD method applied to a neuronal population, we apply this method to several examples of neuronal populations and other biological oscillatory models. We find that our method can be used to reproduce complex tissue-level behaviors such as bifurcations and clustering. We also extended the method to non-identical units whose dynamics vary with a parameter. Using this extension, we studied the synchronization of non-homogeneous neurons with varying periods.

I.3: Overview of Chapter IV

This chapter presents a software framework for simulating a large neuronal network and its application to simulate the cortex of a mouse. The coding and modeling is joint work with James Hazelden, with some help from Dr. Tomoyuki Mano from Hiroki Ueda's group. The modeling of the mouse brain is based on the CUBIC-Atlas [MMS+18] from their group.

The computational software is implemented using `CUDA` to leverage the graphics processing unit (GPU) for computing. We can simulate all neurons present in the mouse cortex in the CUBIC-Atlas with approximately 13 million neurons and around 1 billion synaptic connections on a high-end desktop computer with an NVIDIA RTX 3090 GPU. Furthermore, we generated the connectivity between neurons. We classified the neurons as excitatory or inhibitory in an experimental data-driven method by integrating high-resolution models of connectivity [OHN+14, EGKM18] and distribution of excitatory versus inhibitory neurons [LHA+07, KYP+17] with the coordinate locations provided by the CUBIC-Atlas. Using this software framework accompanied with a realistic connectivity and morphology model, we examined the formation of waves, determined functional connectivity between different cortex regions, and demonstrated the model's ability to generate Electroencephalogram (EEG) traces.

I.4: Overview of our Contribution

I.4.1: Numerical methods

Models for a single neuron ranges from the integrate-and-fire (IF) neurons to multi-compartment models that simulate propagation along axon and dentrite [HAH+11]. There already exist many simulation methods for neurons, each with its own tradeoffs. Be it between the complexity of a single neuron versus the size of the population or balancing the performance versus flexibility in the implementation.

If the connectome of the network is essential in the simulation goal, then the APPD method or any method based on the population density would not be sufficient on its own since the assumption of a population density method discards the detailed connectivity between neurons. Many frameworks simulate a neuronal network, which could be on a supercomputer [Mar06], on a field-programmable gate array (FPGA) [YDW+19], on a workstation with GPU [YTN16] (where we focus) or CPU [GB08]. For simulating 1 million neurons, [YDW+19] reports real-time simulation on FPGA, whereas [YTN16] is 4 times slower than real-time when simulated on a GPU. Our method has its performance comparable to the latter. It should be noted that when the size of the network increases, the real-time simulation property of the FPGA is retained, while the GPU will take a longer time. However, the performance advantages come with the tradeoff of using dedicated hardware and difficulties in programming. Simulation on a GPU does not have any limit in modeling when compared to simulation on a CPU, though the performance gain is modest, [YTN16] reports only a 4-times performance gain over a single-threaded CPU simulation when using single precision for what they claim as *a more realistic setting*, and the performance gain is less when double precision is used. It may be possible that a carefully implemented parallel algorithm on a CPU could be faster, given the number of cores on a single desktop CPU can be up to 16 on an `AMD Ryzen 9 5950X`, and the workload is close to perfectly parallel. In [NSS+16], a comparison for simulation of a recurrent neural network between GPU and CPU is conducted, and a CPU-based simulation has comparable or better performance with a GPU-based version when both are results from optimized software for the platform (Intel MKL for CPU, CuBLAS for GPU). However, [GB08], as an example, is implemented in Python that takes more tradeoff in performance, and they reported two to four times slower than a single-threaded C implementation. Our neuronal network method performs comparably to state-of-the-art competitions that can be achieved on a desktop computer.

The APPD method serves a different purpose. If the neuron network is large and macroscopic properties such as the average membrane potential is of interest, then the APPD

method can be applied. While further simplifications such as a neural field model can still be used to simulate EEG [HSBL18], they did not simulate the ionic channels of individual neurons. We will demonstrate that the APPD method accurately simulates the 4-dimensional Hodgkin-Huxley model, whereas previous population-density-based methods are mostly restricted to 1 or 2-dimensional oscillators [OA08, NK10, PM14]. The ability to use a biophysically realistic model without simplification establishes the correspondence between a change in parameters of the neuronal model to the macroscopic properties of a population.

Additionally, the APPD method can use the same neuron model as the neuronal network simulation, where the only difference is the morphology (locations of the neurons) and the connectivity between neurons. By comparing the results between the models, a user can determine if the behavior observed is likely due to the parameters chosen or the connectivity network between the neurons, based on whether this behavior is present in both simulation results. The APPD method can be used to quickly search through different parameters of the neuronal model that generates a macroscopic behavior, such as a firing pattern seen during sleep. When the appropriate parameters are determined, the additional connectivity and morphology information can be included to allow the study of spatial patterns such as waves and the effect of connectivity between regions.

The Chapter II discussed the LSKF, which is an extension to the Kalman filter. In [SF16], a particle method is introduced to solve the convection-diffusion equation, which takes the same form as a Fokker-Planck equation. We would like to improve the performance of this particle method by using multivariate normals as the kernel function for each particle. Searching the literature, we noticed that the convection-diffusion equation shares the same form of the Fokker-Planck equation formulation, which is present in the *time-update* step of the continuous-discrete Kalman filtering process. We first surveyed the literature in the field of Kalman filtering, as the Kalman filtering process a local linear approximation and gave a deterministic method to simulate noise by deforming the Gaussian; both are properties

desired for the particle. Specifically, we considered the Unscented Kalman Filter and its variations [JU97, JU04, Sar07] and the Continuous-Discrete Cubature Kalman filter [AH09, AHH10], but find them not satisfactory for our application, with two primary limits. First, they require the Jacobian of the model equations. Requiring the explicit Jacobian formula for all models would prevent easy adaptation to more complicated models, while a quadrature rule is numerically unstable, making the method less robust. Second, previous methods used the Ito-Taylor expansion and a pre-determined choice of timestep discretization for the diffusion process. We then considered this problem by tracking a level set of the Gaussian instead to avoid the problems mentioned, as the method does not require higher derivatives and works with standard adaptive ordinary differential equation (ODE) solvers. Since both problems share the same underlying differential equation, we adapted our method to the continuous-discrete Kalman filtering process and evaluated its performance against the state-of-the-art methods. The result of this adaption and comparison is presented in Chapter II.

I.4.2: Modeling: a Data-driven High-Resolution Connectivity Model

Previously, the neuronal atlas for neurons in a mouse brain is limited to labeled regions that do not resolve single neurons [Don08, JBB+10], and many connectivity models are also limited to probabilistic models between those labeled regions [OHN+14, YB16], limiting their spatial resolution. Consequently, existing whole cortex models [Mar06, MWJB17] are also limited by this deficiency. In [MWJB17], their mouse brain model is based on a macroscopic neuronal mass model, which does not resolve individual neurons. The CUBIC-Atlas [MMS+18] is the first neuronal atlas that maps the coordinates of single cells. Using it as the background, we created a data-driven neuronal connectivity model by integrating additional experiment-based atlases. We constructed the neuron connectome between regions of the mouse cortex using the *voxel-based* connectivity model proposed in [KHG+19] and determined the distribution based on experimental data [EGKM18] as well.

This develops the first whole-cortex model that can resolve single neurons in the brain

with a high-resolution experimental-based connectome.

CHAPTER II

The Level Set Kalman Filter

II.1: Introduction

Kalman Filtering methods are used in many applications. A Bayesian filtering method updates a *state estimation* of an object in its state space given knowledge of the system and measurements[Sär13]. The goal of these methods is to estimate the state of a target system for which the dynamics and the statistic of the noise are known, using measurements taken at a fixed time intervals, and accounting for noise or uncertainty in the system and measurements. There are two parts to these methods. First, a new measurement is used to generate the best possible estimate of the system state in the *measurement-update* step. Second, that estimate is propagated forward in time using the system's dynamics till the next measurement is available in the *time-update* step. Here, we present a method for accurately implementing that second step in the presence of noise.

The general framework for these problems was first described by Kalman [KB61]. A Kalman-Bucy type filtering consists of two steps: a *measurement-update* part that updates the estimation using the measurement at, a state estimation from previous steps, and a *time-update* step that updates the state estimation between consecutive measurements. The level set Kalman filter (LSKF) method focuses on the improvement of the state estimation in the *time-update* step, and the discussions that follow are restricted to the *time-update* part unless we explicitly mention the *measurement-update*.

Assuming that the dynamics are linear (in space), and all noise is zero-mean Gaussian, Kalman-Bucy filtering [KB61] gives an optimal way to estimate the system state for the *time-update*. However, in many cases, we would like to generalize this method to a system for which the dynamics of interest are nonlinear. For such a nonlinear system, a Gaussian probability density function (PDF) is no longer preserved even when the dynamics are quadratic [GH11]. When the dynamics are approximately linear for the region of state space where most of the PDF lies, Unscented Kalman filtering (UKF) [JU04] can provide a useful method. Additionally, the UKF is easy to implement since it does not require explicit evaluation of the Jacobian of the velocity field, which is not readily available in practical problems where, for example, the velocity field is implicitly defined as a solution to an equation.

Researchers have improved how the process noise is incorporated, but so far, methods are significantly more complicated than the UKF (e.g., requiring the explicit calculation of a Jacobian) or only work with specific numerical solvers. For example, the continuous-discrete Kalman filter[Sar07] (CDKF) and the continuous-discrete Cubature Kalman filter[AHH10] (CD-CKF). (Note the latter uses the Cubature Kalman transformation as introduced in [AH09] instead of the unscented Kalman transformation, however it can be reformulated to use either, as explained in [KK17].) The CDKF in [Sar07] addresses the continuous nature of the process noise; however, the derivation of their method involves approximation such that their method is not exact even if the dynamics are linear. Moreover, the computation of the prediction is significantly more complicated than the original UKF, eroding the advantage the CDKF offers by removing intermediate timesteps. The CD-CKF uses a 1.5-order (order 1 strong convergence, and order 2 weak convergence) Ito-Taylor expansion of the stochastic differential equation, which uses the Jacobian (or approximations of it) that can be difficult to calculate. Though the explicit Jacobian can be avoided by deriving specific Runge-Kutta methods as described in [New91], this still complicates programming and limits the type of numerical solvers available.

Here, we propose the LSKF that addresses these issues. Our method: 1) does not require

the Jacobian or any spatial partial derivative of the drift function explicitly, 2) allows the use of adaptive ordinary differential equation(ODE) solvers and frees the user from choosing the time discretization, and 3) shows performance improvements over CD-CKF, even in the challenging test cases presented in [AHH10]. From a theory point of view, our derivation of the method is based on the apparent velocity (Remark: the word *apparent* is used to distinguish that it is the velocity of an abstract set and not the object we are tracking.) of the level set of the probability distribution, which is a novel approach to analyze these problems, and may enable further developments.

II.2: Problem Statement and Background

(Note on notation: we distinguish matrix or vector-valued quantities $\mathbf{v}$ versus scalar-valued quantities v_1 by using a bold font.)

II.2.1: Problem formulation

A continuous dynamic discrete measurement system includes a continuous-time process described by a Fokker-Planck equation and a discrete measurement process with measurement noise.

The discrete measurement process is defined by a transformation h from state space to the observation space, together with a zero-mean Gaussian observation noise $\boldsymbol{\tau} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. Suppose at the time of measurement, the state vector is $\mathbf{x}$, then the measurement $\mathbf{y}$ is given by:

$$\mathbf{y} = h(\mathbf{x}) + \boldsymbol{\tau}, \tag{II.2.1}$$

where $\boldsymbol{\tau} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$.

In between the time where two consecutive measurements are taken, we assume that the

process noise is Gaussian, and the system equations are described by the Itô process [Jaz07]:

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v}(\mathbf{x}) + \sqrt{\mathbf{K}}\frac{\partial \beta}{\partial t}, \tag{II.2.2}$$

where $\mathbf{v}$ is the **drift function**, or velocity field defined by the dynamics, β is a standard d dimension Brownian process. $\mathbf{K}$ is an $d \times d$ positive semi-definite continuous **process noise matrix**, and $\mathbf{K} = \sqrt{\mathbf{K}}\sqrt{\mathbf{K}}^T$.

Then, the PDF u is described by the Fokker-Planck equation of the following form:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{v}u). \tag{II.2.3}$$

II.2.2: Brief review of existing time-update methods

Under the assumption that the drift function $\mathbf{v}$ is linear in space and the process noise matrix $\mathbf{K}$ is constant, it can be shown that a Gaussian PDF u is preserved. (A proof of this fact using level sets is in the next section). In [KB61], the derivation of the *time-update* step is based on this observation.

One often wants to generalize this method to nonlinear models even when Gaussian distributions are no longer exactly preserved. One generalization would be to use the Jacobian of the drift function at the mean of the distribution, which is a key part of the Extended Kalman-Bucy Filter (EKF) method. One disadvantage of the EKF is the need for an explicit formula of the Jacobian of the drift function. The UKF is also derived based on the assumption of a local linearization of velocity; however, the explicit evaluation of the Jacobian is avoided.

In [AHH10], after their comparison between the continuous-discrete cubature Kalman filter (CD-CKF), continuous-discrete unscented Kalman filter (CD-UKF), and continuous-discrete extended Kalman filter (CD-EKF), they concluded that *the CD-CKF is the choice*

for challenging radar problems. In [KK17], Kulikov and Kulikova presented a new filtering method named the accurate continuous-discrete extended Kalman filter (ACD-EKF), and compared it to the CD-CKF and CD-UKF. Note the implementation of the CD-UKF in [KK17] is more sophisticated than that in [AHH10] as it uses the Ito-Taylor expansion of order 1.5 (IT-1.5) that is the same as presented in [AHH10] for the CD-CKF. With the improved implementation of the CD-UKF, Kulikov and Kulikova reported in [KK17] that the CD-CKF and the CD-UKF perform similarly. In addition, while the ACD-EKF requires less tuning than the CD-CKF, with sufficient timestep subdivision, the CD-CKF seems to outperform the ACD-EKF, as stated in the conclusion of [KK17]: *The highest accuracy is provided by the most time-consuming filters CD-CKF256 and CD-UKF256.* Therefore, we conclude that with sufficient timestep subdivision, the CD-CKF is still a benchmark method to compare against.

II.2.3: the time-update of CD-CKF with Ito-Taylor expansion

In [AHH10], the IT-1.5 is first introduced to the *time-update* step of the continuous-discrete filtering. It is confirmed in [SS12] that the Unscented Kalman filtering with IT-1.5 achieves similar performance to the CD-CKF. For the purpose of comparing time-update, the performance of the CD-CKF should suffice for a benchmark. Additionally, we noted that while the IT-1.5 should converge to the accurate result with a weak order of convergence 2, the implementation in [AHH10] chooses to only discretize noise once between the measurements and does not converge to this result, presumably as a tradeoff to improve speed. For the sake of complete comparison, we also implemented a version with a proper IT-1.5 expansion that discretizes noise for every timestep subdivision.

Here we restate the *square-root form* of the CD-CKF, as derived in [AHH10].

Time-update: For update with a timestep of Δt, we define the function

$$\mathbf{f}_d(\mathbf{x}, t) := \mathbf{x}(t) + \Delta t \mathbf{v}(\mathbf{x}(t), t) + \frac{1}{2}\Delta t^2(\mathbb{L}_0(\mathbf{v}(\mathbf{x}, t))), \tag{II.2.4}$$

where the opertor

$$\mathbb{L}_0 := \frac{d}{dt} + \sum_{i=1}^{d} v_i \frac{\partial}{\partial x_i} + \frac{1}{2} \sum_{j,p,q=1}^{d} \sqrt{\mathbf{K}}_{p,j} \sqrt{\mathbf{K}}_{j,q} \frac{\partial^2}{\partial x_p \partial x_q}. \tag{II.2.5}$$

We also define the operator $\mathbb{L}(\mathbf{v})$ as the square matrix defined entrywise with its (i,j)th element being $\mathbb{L}_j v_i$, where

$$\mathbb{L}_j := \sum_{i=1}^{d} \sqrt{\mathbf{K}}_{i,j} \frac{\partial}{\partial x_i}. \tag{II.2.6}$$

Then the time-update algorithm is as follows:

Require: Guess of initial state $\mathbf{x}_0$ at time t_0, and a factorization of a guess of covariance matrix $\mathbf{M}$. Drift velocity $\mathbf{v}$, continuous process noise matrix $\mathbf{K}$.

1: Find the $2d$ cubature points $\mathbf{x}_i = \mathbf{x}_0 + (\mathbf{M})_i, \mathbf{x}_{i+d} = \mathbf{x}_0 - (\mathbf{M})_i$, where $i = 1, \ldots, d$, and $(\mathbf{M})_i$ denotes the ith column of $\mathbf{M}$.

2: Evaluate the propagated cubature point: $\mathbf{x}_i^* = \mathbf{f}_d(\mathbf{x}_i, t_0)$, where where $i = 1, \ldots, 2d$.

3: Estimate the updated mean by the average of the cubature points:

$$\mathbf{x}_0^* := \frac{1}{2d} \mathbf{x}_i^*. \tag{II.2.7}$$

4: Estimate the factorization of covariance matrix by the triangularization of the concatenated matrix:

$$\mathbf{M}^* = \text{tria}\left(\left[\mathbf{X}^* | \sqrt{\Delta t} (\sqrt{\mathbf{K}} + \tfrac{\Delta t}{2} \mathbb{L}(\mathbf{v})) | \sqrt{\tfrac{\Delta t^3}{12}} \mathbb{L}(\mathbf{v}) \right] \right), \tag{II.2.8}$$

where tria$(\cdot)$ denotes applying a triangularization procedure such as the Gram-Schmidt based QR-decomposition, $\mathbf{X}^*$ is a matrix with ith column being $\mathbf{x}_i^*$, $\mathbb{L}(\mathbf{v}) = \mathbb{L}(\mathbf{v}(\mathbf{x}_0^*, t_0))$.

return Estimated updated mean $\mathbf{x}_0^*$ and updated factorization of covariance matrix $\mathbf{M}^*$ at $t_0 + \Delta t$.

II.2.4: The square root form of cubature Kalman measurement-update

Here, we discuss the *measurement-update* method used in the CD-CKF and the LSKF. Since the operations from the time-update can cause $\mathbf{M}$ to be positive semi-definite, a measurement-update method that can accommodate a positive semi-definite matrix is required for reliability, as pointed out in [AHH10]. We used the measurement-update method from the square root CD-CKF method, as stated in Appendix B of [AHH10]. Since the notations used are different, the measurement-update of the square root CD-CKF is restated here for reference.

Require: Factorization of the predicted covariance matrix before measurement $\mathbf{M}$, predicted mean before measurement $\bar{\mathbf{x}}$, measurement $\mathbf{y}$, a factorization of the covariance of the measurement noise matrix $\sqrt{\mathbf{R}}$, measurement function $\mathbf{h}$

1: Find the concatenated cubature points matrix of size $d \times 2d$:

$$\mathbf{N} = \bar{\mathbf{x}} + \sqrt{2d}\,[\mathbf{M}| - \mathbf{M}]\,, \tag{II.2.9}$$

where the vector-matrix addition is applied as $\bar{\mathbf{x}}$ added to each **column** of the concatenated matrix $[\mathbf{M}| - \mathbf{M}]$

2: Evaluated the propagated cubature points

$$\mathbf{Y} = \mathbf{h}(\mathbf{N}), \tag{II.2.10}$$

where the measurement function $\mathbf{h}(\cdot)$ is evaluated on each **column**.

3: Estimate the predicted measurement

$$\bar{\mathbf{y}} = \frac{1}{2d}\sum_{i=1}^{2d}\mathbf{Y}_i. \tag{II.2.11}$$

4: Compute matrices $\mathbf{T}_{11}$,$\mathbf{T}_{21}$, and $\mathbf{T}_{22}$ by the following QR-factorization:

$$\begin{bmatrix}\mathbf{T}_{11} & \mathbf{O}\\ \mathbf{T}_{21} & \mathbf{T}_{22}\end{bmatrix} = \mathrm{qr}\left(\begin{bmatrix}\mathbf{Y} & \sqrt{\mathbf{R}}\\ \mathbf{N} & \mathbf{O}\end{bmatrix}\right), \tag{II.2.12}$$

where $\mathbf{O}$ denotes a zero matrix of appropriate size.

5: Estimate the cubature gain

$$\mathbf{W} = \mathbf{T}_{21}/\mathbf{T}_{11}, \tag{II.2.13}$$

where / represents solving for $\mathbf{W}$ in $\mathbf{T}_{21} = \mathbf{W}\mathbf{T}_{11}$ using a backward stable solver.

6: Estimate the mean of the corrected state

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{W}(\mathbf{y} - \bar{\mathbf{y}}). \tag{II.2.14}$$

7: Estimate a factorization of the corrected covariance matrix

$$\hat{\mathbf{M}} = \mathbf{T}_{22}. \tag{II.2.15}$$

return Corrected mean $\hat{\mathbf{x}}$ and a factorization of the corrected covariance matrix $\hat{\mathbf{M}}$.

II.3: Derivation of the time-update of the level set Kalman filter

In this section, we focus on deriving the *time-update* of the level set Kalman filter (LSKF). In the first subsection, we show that a Gaussian is preserved by a local linear approximation to the original Fokker-Planck equation by tracking its level set. In this process, we observe that

the apparent velocity of the level set is given by the drift function plus an additional term which we name as the **diffusion velocity**. In the second subsection, using the apparent velocity of the level set, we derive a numerical method that tracks such Gaussian particles for the *time-update* step. In the third subsection, we state the **averaged velocity** version of the *time-update* part of the LSKF, which turns out to give better results numerically.

II.3.1: Preservation of Gaussian for local linear approximation

Without loss of generality (WLOG), we may assume the particle of concern is centered at $\mathbf{0}$. Moreover, since we are interested in how the dynamics and diffusion *deform* the distribution, we may also set the drift function at center $\mathbf{v}(\mathbf{0}) = \mathbf{0}$. With these simplifications in mind, the *original* Fokker-Planck equation can be restated as:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{v}u), \tag{II.3.1}$$

where $u = u(\mathbf{x}, t)$ is the PDF, $\mathbf{K}$ is a constant matrix-valued continuous Gaussian **process noise**, and $\mathbf{v} = \mathbf{v}(\mathbf{x})$ is the **drift velocity (field)**, and $\mathbf{v}(\mathbf{0}) = \mathbf{0}$ by our WLOG simplification.

Then, we approximate the above (II.3.1) by taking a linear approximation of $\mathbf{v}$. In notations, we assume that $\mathbf{v}(\mathbf{x}) \approx \mathbf{J}\mathbf{x}$. ($\mathbf{J}$ is the Jacobian matrix.) Then:

Claim II.1. *A Gaussian distribution is preserved by a Fokker-Planck equation with a linear drift function.*

Stated explicitly: For the following equation:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{J}\mathbf{x}u), \tag{II.3.2}$$

if the initial condition $u(\mathbf{x}, 0)$ is given by a Gaussian function

$$u(\mathbf{x}, 0) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \exp\left(-\frac{\mathbf{x}^T \mathbf{\Sigma}^{-1} \mathbf{x}}{2}\right), \tag{II.3.3}$$

then $u(\mathbf{x}, t)$ is also a Gaussian distribution.

It should be noted that the result is not new, but our proof leads to an explicit formula for updating the density. The rest of this subsection proves this claim.

We define the auxiliary function F by

$$F(\mathbf{x}, t) := \frac{u(\mathbf{x}, t)}{u(\mathbf{0}, t)}. \tag{II.3.4}$$

Consider a **level set** of the function F at t defined as

$$\mathcal{L}(t) := \left\{\mathbf{x} \in \mathbb{R}^d | F(\mathbf{x}, t) = c\right\}, \tag{II.3.5}$$

where $0 < c < 1$ is some fixed scalar constant. $\mathcal{L}(t)$ is (usually) a surface, and for a Gaussian as defined in $u(\mathbf{x}, 0)$, it is an ellipsoid. As the function F varies in time, the set $\mathcal{L}$ propagates in space. To describe the movement of the set $\mathcal{L}$, we consider the apparent velocity the traveling surface.

In particular, a **velocity of level set** $\mathbf{v}_L$ is *defined* by a velocity field satisfying the **level-set equation**:

$$\frac{\partial F}{\partial t} + \mathbf{v}_L \cdot \nabla F = \mathbf{0}. \tag{II.3.6}$$

(Note: this can be understood as the chain rule. For a more detailed explanation, refer to equations (1) and (2) in [Set85]. Also, note the **velocity of the level set** is uniquely defined up to tangential directions since tangential movements along the level set vanish since they preserve the level set.)

To proceed to the proof, we first consider the lemma:

Lemma II.2. *The velocity field*

$$\mathbf{v}_L = \mathbf{J}\mathbf{x} + \frac{1}{2}\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x} \tag{II.3.7}$$

is a velocity of the level set for $\mathcal{L}(0)$ defined in (II.3.5). Also, this velocity of the level set is linear in space.

Proof of Lemma II.2. First, we note the velocity field is linear in space. To check that it is a velocity of the level set:

Since $F(\mathbf{x}, t)$ is defined as a quotient of $u(\mathbf{x}, t)$ and $u(\mathbf{0}, t)$, we may omit the normalizing factor in u, and take

$$u(\mathbf{x}, 0) = \exp\left(-\frac{\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}}{2}\right) \tag{II.3.8}$$

as the initial condition. Then:

$$\frac{\partial F}{\partial t}|_{t=0} = \frac{u'(\mathbf{x}, 0)u(\mathbf{0}, 0) - u'(\mathbf{0}, 0)u(\mathbf{x}, 0)}{u^2(\mathbf{0}, 0)}. \tag{II.3.9}$$

By (II.3.8), we note that $u(\mathbf{0}, 0) = 1$. We simplify (II.3.9) using this substitution:

$$\frac{\partial F}{\partial t}|_{t=0} = u'(\mathbf{x}, 0) - \exp\left(-\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}\right) u'(\mathbf{0}, 0). \tag{II.3.10}$$

Substitute time derivatives u' with (II.3.2):

$$\begin{aligned} \frac{\partial F}{\partial t}|_{t=0} = &\left(\frac{1}{2}\nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{J}\mathbf{x}u)\right)|_{t=0,\mathbf{x}=\mathbf{x}} \\ &- \exp(\dots)\left(\frac{1}{2}\nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{J}\mathbf{x}u)\right)|_{t=0,\mathbf{x}=\mathbf{0}}, \end{aligned} \tag{II.3.11}$$

where

$$\exp(\dots) := \exp\left(-\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}\right). \tag{II.3.12}$$

continuing the computation, we find that:

$$\nabla \cdot \mathbf{K}\nabla u = \nabla \cdot \left(-\mathbf{K}\exp(\dots)\boldsymbol{\Sigma}^{-1}\mathbf{x}\right) \tag{II.3.13}$$

$$= \exp(\dots)\boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x} - \exp(\dots)\operatorname{tr}(\mathbf{K}\boldsymbol{\Sigma}^{-1}) \tag{II.3.14}$$

$$= \exp(\dots)\left(\boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x} - \operatorname{tr}(\mathbf{K}\boldsymbol{\Sigma}^{-1})\right). \tag{II.3.15}$$

And

$$\nabla \cdot ((\mathbf{J}\mathbf{x}u)) = \nabla u \cdot \mathbf{J}\mathbf{x} + u\nabla\cdot\mathbf{J}\mathbf{x} \tag{II.3.16}$$

$$= \exp(\dots)(-\boldsymbol{\Sigma}^{-1}\mathbf{x})\cdot\mathbf{J}\mathbf{x} + \exp(\dots)\operatorname{tr}(\mathbf{J}) \tag{II.3.17}$$

$$= \exp(\dots)\left(\operatorname{tr}(\mathbf{J}) - \boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{J}\mathbf{x}\right). \tag{II.3.18}$$

Substitute these two terms into (II.3.11), we have

$$\begin{aligned}\frac{\partial F}{\partial t} &= \exp(\dots)\left(\frac{1}{2}\boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\operatorname{tr}(\mathbf{K}\boldsymbol{\Sigma}^{-1})\right. \\ &\quad \left. +\ \operatorname{tr}(\mathbf{J}) - \boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{J}\mathbf{x}\right) \\ &\quad - \exp(\dots)\left(-\frac{1}{2}\operatorname{tr}(\mathbf{K}\boldsymbol{\Sigma}^{-1}) + \operatorname{tr}(\mathbf{J})\right)\end{aligned} \tag{II.3.19}$$

$$= \exp(-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x})\left(\frac{1}{2}\boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\Sigma}^{-1}\mathbf{x}\cdot\mathbf{J}\mathbf{x}\right). \tag{II.3.20}$$

Meanwhile, we check that

$$\mathbf{v}_L\cdot\nabla F = \mathbf{v}_L\cdot\nabla\left(\exp(-\frac{\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}}{2})\right) \tag{II.3.21}$$

$$= \mathbf{v}_L\cdot\left(-\boldsymbol{\Sigma}^{-1}\mathbf{x}\exp(-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x})\right). \tag{II.3.22}$$

Substitute the level set velocity term $\mathbf{v}_L$ from (II.3.7), we get that

$$\mathbf{v}_L \cdot \nabla F = -\exp(-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x})(\mathbf{J}\mathbf{x} + \frac{1}{2}\mathbf{K}\boldsymbol{\Sigma}^{-1}\mathbf{x}) \cdot \boldsymbol{\Sigma}^{-1}\mathbf{x}. \tag{II.3.23}$$

Comparing the results from (II.3.23) and (II.3.20), we conclude that

$$\frac{\partial F}{\partial t} + \mathbf{v}_L \cdot \nabla F = 0. \tag{II.3.24}$$

Therefore $\mathbf{v}_L$ is a velocity of the level set, as defined in (II.3.6). □

We now proceed to the proof Claim II.1:

Proof of Claim II.1. Lemma II.2 shows that every level set is propagated by a linear velocity field independent of the choice of level set (in other words, independent of the choice of c.) In particular, it is propagated by a linear transformation instantaneously. Consequently, between any fixed time 0 and Δt, the Gaussian is mapped by a linear transformation. Since a linear transformation maps Gaussians to Gaussians, the velocity field $\mathbf{v}_L$ is always well-defined as the covariance term in (II.3.7) is defined, whereas other terms are known. □

Before we proceed, we should note that the claim is a corollary of equation (29) in [KB61] by Kalman and Bucy that started the discussion of continuous-discrete Kalman filtering. The significance of the proof is that by using a square-root factorization, the transformation is now given by an *explicit formula* (II.3.7) that does not involve an integral (An example of such a formula is (20) in [KB61]).

II.3.2: Deriving the time-update of the LSKF

We now describe the numerical algorithm inspired from the velocity of level set (II.3.7). Tracking the movement of the Gaussian is equivalent to tracking one of its ellipsoid level sets (as defined in (II.3.5)). If the mean of the Gaussian remains at $\mathbf{0}$, then a factorization of the covariance matrix $\boldsymbol{\Sigma} = \mathbf{M}\mathbf{M}^T$ can be used to represent the Gaussian. This factorization

also represents the unique level set ellipsoid spanned by the columns of the factorization $\mathbf{M}$. More specifically, set

$$\mathbf{M}(0) = \begin{bmatrix} \mathbf{x}_1(0) & \cdots & \mathbf{x}_d(0) \end{bmatrix} \tag{II.3.25}$$

as initial conditions, and let $\mathbf{x}_i(t)$ be the solutions of (II.3.7). (One may interpret $\mathbf{x}_i(t)$ as a **point on the level set**, which travels at the apparent speed defined by a velocity of level set.)

Then $\mathbf{\Sigma}(t)$ defined as

$$\mathbf{\Sigma}(t) := \mathbf{M}(t)\mathbf{M}(t)^T \tag{II.3.26}$$

is the covariance matrix for the Gaussian at time t since it is a similarity transformation. Suppose A is the linear transformation from time 0 to t, then $\mathbf{x}_i(t) = \mathbf{A}\mathbf{x}_i(0)$, and

$$\begin{aligned} \mathbf{\Sigma}(t) &= \mathbf{M}(t)\mathbf{M}(t)^T \\ &= \mathbf{A}\mathbf{M}(0)\mathbf{M}(0)^T\mathbf{A} \\ &= \mathbf{A}\mathbf{\Sigma}(0)\mathbf{A}^T. \end{aligned}$$

Therefore applying linear transformation $\mathbf{A}$ to the ellipse is equivalent to applying it to all column vectors in $\mathbf{M}$.

The Jacobian of the velocity field $\mathbf{J}$ that appears in (II.3.7) is not explicitly needed. Instead of direct evaluation of the Jacobian, we approximate the effect of the drift velocity by applying a quadrature rule. In this section, we use the forward difference in space to derive the method. Specifically, we notice $\mathbf{\Sigma}^{-1} = \mathbf{M}^{-T}\mathbf{M}^{-1}$, (where $\mathbf{M}^{-T}$ indicates the inverse transpose) and the $\mathbf{x}_i$s are columns of $\mathbf{M}$. Therefore for all $\mathbf{x}_i$' on the level set ellipsoid, by substituting the terms in (II.3.7), we find the apparent velocity of the level set

is approximated by

$$\frac{\partial \mathbf{x}_i}{\partial t} = \mathbf{v}(\bar{\mathbf{x}} + \mathbf{x}_i) - \mathbf{v}(\bar{\mathbf{x}}) + \frac{1}{2}\mathbf{K}(\mathbf{M}^T)^{-1}\mathbf{e}_i, \tag{II.3.27}$$

where $\bar{\mathbf{x}}$ is the **mean** of the Gaussian. In the case the Gaussian is centered at $\mathbf{0}$, $\bar{\mathbf{x}} = \mathbf{0}$. $\mathbf{e}_i$ is the ith unit vector with all entries 0 except that ith entry is 1.

Recall that the $\mathbf{x}_i$s are columns of $\mathbf{M}$. In a matrix short-hand (where the matrix-vector additions are defined entry-wise, and recall vectors are column vectors):

$$\frac{\partial \mathbf{M}}{\partial t} = \mathbf{v}(\bar{\mathbf{x}} + \mathbf{M}) - \mathbf{v}(\bar{\mathbf{x}}) + \frac{1}{2}\mathbf{K}(\mathbf{M}^T)^{-1}. \tag{II.3.28}$$

Whereas the velocity of the mean value is given by

$$\frac{\partial \bar{\mathbf{x}}}{\partial t} = \mathbf{v}(\bar{\mathbf{x}}). \tag{II.3.29}$$

Based on the form of the equations, the assumptions $\bar{\mathbf{x}} = \mathbf{0}$ and $\mathbf{v}(\bar{\mathbf{x}}) = \mathbf{0}$ can be dropped.

Concatenating $\bar{\mathbf{x}}$ and $\mathbf{M}$ as a variable $(\bar{\mathbf{x}}|\mathbf{M})$ of dimension $d \times (d+1)$, we obtained a nonlinear ODE in this space. Any standard ODE solver can be applied to this ODE to complete the *time-update* between the measurements.

Note that to evaluate the velocity of one point $\mathbf{x}_i$ on the level set, both the mean $\bar{\mathbf{x}}$ and all other points $\mathbf{x}_j$ for this Gaussian kernel are needed, hence the points on a level set cannot be updated independently (in contrast to the time-update step for both UKF and CD-CKF). This provides intuition about the difference between our method and others: while other methods look at the past covariance information and rely on an expansion in time, our method uses only the current information about the covariance matrix. Except for the purpose of numerically solving the ODE, our method does not need time-discretization.

II.3.3: Motivating example: linear drift function

As an illustration for the *time-update* method, we consider the following Fokker-Planck equation with a linear drift function:

$$\frac{du}{\partial t} = \nabla \cdot \mathbf{K}\nabla u - \nabla \cdot (\mathbf{J}\mathbf{x}u) \tag{II.3.30}$$

with parameters

$$\mathbf{K} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{2} \end{bmatrix} \quad \mathbf{J} = \begin{bmatrix} 0 & 0.1 \\ 0 & 0 \end{bmatrix}.$$

We consider the solution of the initial value problem with initial condition

$$u(\mathbf{x}, 0) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma}_0)}} \exp(-\frac{\mathbf{x}^T \mathbf{\Sigma}_0^{-1} \mathbf{x}}{2}), \tag{II.3.31}$$

where the initial covariance is given by

$$\mathbf{\Sigma}_0 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{II.3.32}$$

To verify the accuracy of our method, we check the convergence of numerical ODE solvers for the initial value problem and find the error of the density function at $t = 10$ with ODE solvers of a different order. The $\mathbf{\Sigma}_{10}$ is solved numerically with sufficient timesteps for computing the convergence. We factor $\mathbf{\Sigma}_0 = \mathbf{M}_0\mathbf{M}_0^T$, and set the ODE (II.3.28) with the initial condition $\mathbf{M}(0) := \mathbf{M}_0$. To verify that our method is accurate for the linear Fokker-Planck equation (II.3.30), we check that when using different numerical ODE solvers, the solution converges to the same value, with the rate of convergence coinciding with the order of the ODE solver.

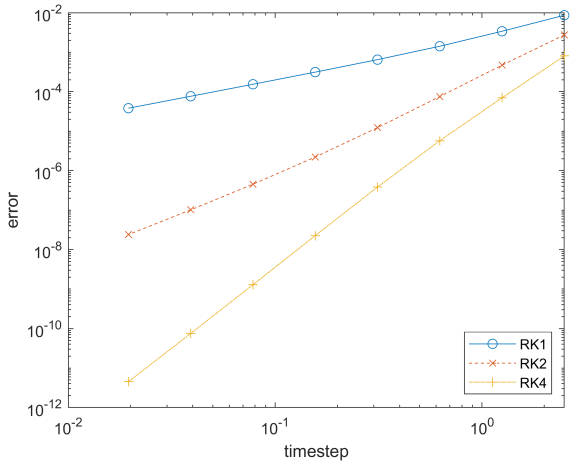


Figure II.1: log-log graph of the error of covariance matrix in the infinity norm with linear Fokker-Planck equation. This shows that our method preserves the order of accuracy of the ODE solver. The least-square fitting of the curve gives the slope 1.03 for RK1, 2.11 for RK2, and 3.85 for RK4, which approximately matches the expected order of convergence 1, 2, and 4 respectively.

In Fig. II.1, we verify that for the Runge-Kutta methods of order 1, 2, and 4, the error of $u(\cdot, 10)$ measured in the infinity norm converges at the same order of the ODE solvers, which is expected if the reformulation (II.3.28) is exact for (II.3.30).

II.3.4: The averaged velocity level set time-update

Here we state the **averaged velocity level set time-update** method, which uses central difference instead of forward difference, and shows better accuracy in numerical experiments (See Appendix A for an example) when compared versus the version in Sec. II.3.2.

We set the velocity of the mean $\partial\bar{\mathbf{x}}/\partial t$ by the **averaged velocity**:

$$\frac{\partial \bar{\mathbf{x}}}{\partial t} = \mathbf{v}_a(\bar{\mathbf{x}}, \mathbf{M}) := \frac{1}{2d} \sum_{i=1}^{d} \left(\mathbf{v}(\bar{\mathbf{x}} + \mathbf{x}_i) + \mathbf{v}(\bar{\mathbf{x}} - \mathbf{x}_i) \right). \tag{II.3.33}$$

(Recall that $\mathbf{x}_i$ are columns of the matrix $\mathbf{M}$)

and the velocity of the matrix $\mathbf{M}$:

$$\frac{\partial \mathbf{M}}{\partial t} = \mathbf{v}(\bar{\mathbf{x}} + \mathbf{M}) - \mathbf{v_a} + \frac{1}{2}\mathbf{K}(\mathbf{M}^T)^{-1}. \tag{II.3.34}$$

It can be easily seen that when the drift velocity field $\mathbf{v}$ is linear in space, equations (II.3.34) and (II.3.28) are identical, and $\mathbf{v}_a(\bar{\mathbf{x}}, \mathbf{M}) = \mathbf{v}(\bar{\mathbf{x}})$.

Using this averaged velocity, here we summarize the LSKF:

Require: Guess of initial state $\hat{\mathbf{x}}_0$ at time t_0, and a factorization of a guess of covariance matrix $\hat{\mathbf{M}}_0$, measurements $\mathbf{y}_1 \ldots, \mathbf{y}_n$ at time $t_1, \ldots, t_n$.

Drift velocity $\mathbf{v}$, continuous process noise matrix $\mathbf{K}$, measurement function h, a factorization of the covariance matrix $\mathbf{R}$ of a zero-mean Gaussian measurement noise.

1: **for** j=1,. . . ,n **do**

2: Set the problem of $\bar{\mathbf{x}}(t)$ and $\mathbf{M}(t)$ given by:

$$\frac{\partial \bar{\mathbf{x}}}{\partial t} = \frac{1}{2d}\sum_{i=1}^{d}\left(\mathbf{v}(\bar{\mathbf{x}} + \mathbf{x}_i) + \mathbf{v}(\bar{\mathbf{x}} - \mathbf{x}_i)\right) \tag{II.3.35}$$

$$\frac{\partial \mathbf{M}}{\partial t} = \mathbf{v}(\bar{\mathbf{x}} + \mathbf{M}) - \mathbf{v_a} + \frac{1}{2}\mathbf{K}(\mathbf{M}^T)^{-1}, \tag{II.3.36}$$

with initial condition $\bar{\mathbf{x}}(t_{j-1}) = \hat{\mathbf{x}}_{j-1}$, and $\mathbf{M}(t_{j-1}) = \hat{\mathbf{M}}_{j-1}$ (Recall: $\mathbf{v_a}$ is defined in (II.3.33), also recall that $\mathbf{x}_i$ are columns of $\mathbf{M}$, and the vector-matrix addition is applied to each column.)

3: *Time-update*: solve the above equation from t_{j-1} to t_j using a numerical ODE solver, and approximate $\bar{\mathbf{x}}_j = \bar{\mathbf{x}}(t_j)$, $\mathbf{M}_j = \mathbf{M}(t_j)$ with the numerical solution.

4: *Measurement-update*: Find the corrected mean $\hat{\mathbf{x}}_j$ and corrected covariance matrix $\hat{\mathbf{M}}_j$ at time t_j by applying the measurement-update algorithm defined in subsection II.2.4, with input $\bar{\mathbf{x}}_j$, $\mathbf{M}_j$, and $\mathbf{R}$.

5: **end for**

return Predicted corrected state $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_n$, at $t_1, \ldots, t_n$, with a factorization of the

predicted corrected covariance matrix $\hat{\mathbf{M}}_1, \ldots, \hat{\mathbf{M}}_n$.

II.3.5: Comparing convergence: achieving beyond IT-1.5 without explicit higher derivatives

In Sec. II.2.3, we introduced the CD-CKF with IT-1.5. Here, we compare the convergence rate of the time-update of CD-CKF (as implemented in [AHH10], and with proper IT-1.5) with that of the LSKF.

Consider the following oscillator:

$$\mathbf{x}(t) := [\epsilon(t) \quad \dot{\epsilon}(t) \quad \ddot{\epsilon}(t)]^T, \tag{II.3.37}$$

where ϵ, $\dot{\epsilon}$, and $\ddot{\epsilon}$ are the position, velocity, and acceleration of the oscillator. Its time derivative is given by

$$\mathbf{v}(\mathbf{x}) = [\dot{\epsilon} \quad \ddot{\epsilon} \quad -\epsilon]^T. \tag{II.3.38}$$

This oscillator is also subject to a continuous process noise, defined by the diagonal diffusion matrix:

$$\mathbf{K} = \mathrm{diag}[0.01^2 \quad 0.01^2 \quad 0.02^2]. \tag{II.3.39}$$

Since the dynamics are linear, the Gaussian is preserved, and we expect the result from LSKF and proper IT-1.5 to converge to the exact solution.

To find the order of convergence, and compare the methods, we consider the following initial condition problem. Given initial condition

$$\mathbf{x}(0) = [1 \quad 0 \quad 0]^T \tag{II.3.40}$$

and initial covariance matrix

$$\mathbf{\Sigma}(0) = \mathrm{diag}[0.01^2 \quad 0.01^2 \quad 0.03^2], \tag{II.3.41}$$

we would like to find the end state at $t = 0.2$ using the above mentioned methods. By subdividing the timesteps, we arrive at the following convergence result:

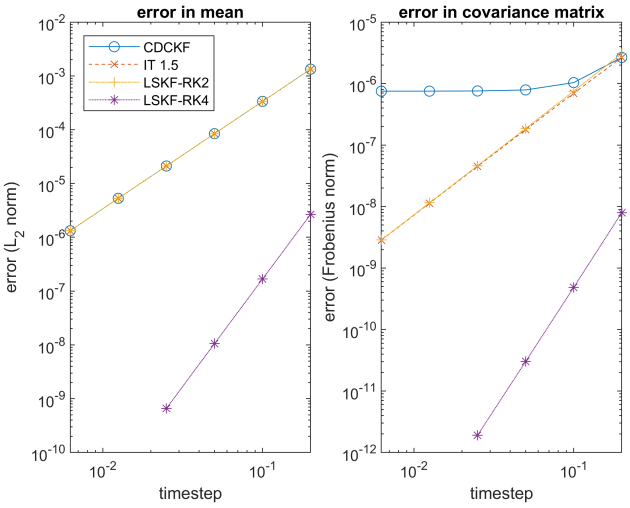


Figure II.2: Convergence of the mean $\mathbf{x}$ and the covariance matrix $\mathbf{\Sigma}$ with the CD-CKF and the LSKF. The left panel shows the error (measured in L_2 norm) in mean value as a function of timestep, whereas the right panel shows error (measured in Frobenius norm) in the covariance matrix. In each panel, CDCKF (blue) denotes the time-update implemented in [AHH10], IT 1.5 (red) denotes the CD-CKF with proper IT-1.5 expansion, LSKF-RK2 (yellow) and LSKF-RK4 (purple) denotes time-update of LSKF with Runge-Kutta solvers of order 2 and 4 respectively. The Runge-Kutta 4 version is cut short due to finite precision linear algebra.

Similar to Fig. II.1, Fig. II.2 shows that the time-update of LSKF converges the same order as the underlying ODE solver. Importantly, note proper IT-1.5 and LSKF-RK2 converge to the same limit mean and covariance matrix at a weak order of convergence 2, validating the correctness of both methods. (Note: they **do not** converge to the same square root of

the covariance matrix, which is not surprising given that the matrix square roots are not unique.) The time-update of CD-CKF as implemented in [AHH10] does not converge to the same limit. Comparing LSKF-RK4 versus proper IT-1.5, it can be noted that much fewer timestep subdivisions can achieve similar truncation errors. As pointed out in [SS12], IT-1.5 already requires explicit first and second derivatives of $\mathbf{v}(\mathbf{x})$, and any higher-order Ito-Taylor expansion is necessarily more complicated. On the contrary, the time-update of LSKF, as defined in (II.3.34) does not require the explicit expression of the derivatives of $\mathbf{v}(\mathbf{x})$, and can achieve a higher order of convergence with the freedom to choose any ODE solver.

II.4: Numerical example: the radar tracking coordinated turn test case

II.4.1: Problem description

Here, we follow the test case presented in [AHH10], considering the scenario where a radar station tracks an aircraft making a coordinated turn. Since the CD-CKF in [AHH10] is claimed to be *the choice for challenging radar problems*, we compare LSKF against CD-CKF in the most challenging scenario they considered with $\omega = 6°/s$ with sampling intervals $T = 2s$, $T = 4s$, and $T = 6s$. (Note: a conversion to radians per second is required.) Additionally, we consider the more challenging scenario with $\omega = 12°/s$ and $\omega = 24°/s$. We implemented the CD-CKF based on the square root form formulated in [AHH10], using their implementation presented at [Ara]. The details of the test case are as follows:

The aircraft is described by a 7-dimensional state vector

$$\mathbf{x}(t) := [\epsilon(t) \quad \dot{\epsilon}(t) \quad \eta(t) \quad \dot{\eta}(t) \quad \zeta(t) \quad \dot{\zeta}(t) \quad \omega(t)]^T, \tag{II.4.1}$$

where $\epsilon(t), \eta(t), \zeta(t)$ are the position, in meters, $\dot{\epsilon}(t), \dot{\eta}(t), \dot{\zeta}(t)$ are the velocity of the aircraft, in meters per second, and the $\omega(t)$ describes the turn rate of the aircraft, in **radians** per

second. The dynamics of the aircraft is defined by the following drift equation:

$$\mathbf{v}(\mathbf{x}(t)) = [\dot{\epsilon} \quad -\omega\dot{\eta} \quad \dot{\eta} \quad \omega\dot{\epsilon} \quad \dot{\zeta} \quad 0 \quad 0]^T. \tag{II.4.2}$$

The noise term is defined by the following diagonal diffusion matrix:

$$\mathbf{K} = \text{diag}([0 \quad \sigma_1^2 \quad 0 \quad \sigma_1^2 \quad 0 \quad \sigma_1^2 \quad \sigma_2^2]), \tag{II.4.3}$$

where $\sigma_1 = \sqrt{0.2}$, and $\sigma_2 = 7 \times 10^{-4}$. (Note: in [AHH10], they suggested $\sigma_2 = 7 \times 10^{-3}$. However, the accompanied code provided by Arasaratnam on his webpage [Ara] used the parameter $\sigma_2^2 = 5 \times 10^{-7}$, which matches closely with $\sigma_2 = 7 \times 10^{-4}$. Our calculated RMSE is similar to the results shown in Fig. 2,3 and 4 in [AHH10] if 7×10^{-4} is chosen, whereas $\sigma_2 = 7 \times 10^{-3}$ does not give similar results.)

The measurement is from a single radar station located at $\mathbf{s} = [1500 \quad 10 \quad 0]$. The radar station measures the distance r, azimuth angle θ and elevation angle ϕ relative to the radar station. The measurement function is therefore given by:

$$\begin{bmatrix} r \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(\epsilon - 1500)^2 + (\eta - 10)^2 + \zeta^2} \\ \arctan(\frac{\eta - 10}{\epsilon - 1500}) \\ \arctan(\frac{\zeta}{\sqrt{(\epsilon-1500)^2 + (\eta-10)^2}}) \end{bmatrix} + \mathbf{w}. \tag{II.4.4}$$

where the measurement noise $\boldsymbol{\tau} \sim \mathcal{N}(0, \mathbf{R})$, with measurement noise matrix $\mathbf{R} = \text{diag}([\sigma_r^2, \sigma_\theta^2, \sigma_\phi^2])$, where $\sigma_r = 50, \sigma_\theta = 0.1^\circ, \sigma_\phi = 0.1^\circ$ (Note: the standard deviations σ_θ and σ_ϕ are measured in **degrees**, and a unit conversion is needed).

For the test scenario, the aircraft starts with the initial state

$$\mathbf{x}_0 = [1000 \quad 0 \quad 2650 \quad 150 \quad 200 \quad 0 \quad \omega_0]^T, \tag{II.4.5}$$

where ω_0 is the initial turn rate, and the measurement is taken with a constant time interval

T. The total time for simulation is chosen to be 120 seconds. The turn rate and measurement interval vary across test cases to examine the performance of the filters. The initial covariance is taken as

$$\mathbf{\Sigma} = \text{diag}([100 \quad 1 \quad 100 \quad 1 \quad 100 \quad 1 \quad 0.01]), \tag{II.4.6}$$

based on a physically realistic assumption: from an observer on the ground, one would have a reasonably good guess about its position with standard deviation $\sigma = 10$ meters, and a good guess about its velocity through differentiation with $\sigma = 1$ meters per second, but a rather bad guess for the turn rate with $\sigma = 0.1$ radian per second, or approximately 5.73 degrees per second.

II.4.2: Numerical results

With the problem description complete, we now turn to present our numerical results. $N = 100$ experiments are executed for each set of parameters, and the same set of experiments is applied to all candidate filters. The main performance metric used is the root-mean square error (RMSE) for position, velocity and turn rate. For example, RMSE for position is defined as:

$$\sqrt{\frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}\left((\epsilon_k^n - \hat{\epsilon}_k^n)^2 + (\eta_k^n - \hat{\eta}_k^n)^2 + (\zeta_k^n - \hat{\zeta}_k^n)^2\right)}, \tag{II.4.7}$$

where N is the number of experiments, and K is the number of measurements in each experiment.

Another metric we consider is the number of divergent results, which we define as any result that has an error larger than 500 or cannot be advanced to the last timestep.

Following [AHH10], we evaluate the performance of each method at different subdivisions, m, of the timestep between measurements. Since our method is defined purely as a reformulated ODE, the subdivision of the timestep is the same as a timestep in a fixed

timestep ODE solver, such as the widely-used Runge-Kutta 4 method. Additionally, to verify our claim that the choice of timestep subdivision can be completely passed to the ODE solver, we also use an adaptive solver, `ode113`, which is integrated into the `MATLAB` software package. In this sense, we introduced 2 implementations of the LSKF, which we call the *LKSF-RK4* and *LSKF-adaptive* respectively. In the following examples, we will verify that the additional subdivisions do not have any effect on the numerical results from the LSKF-adaptive.

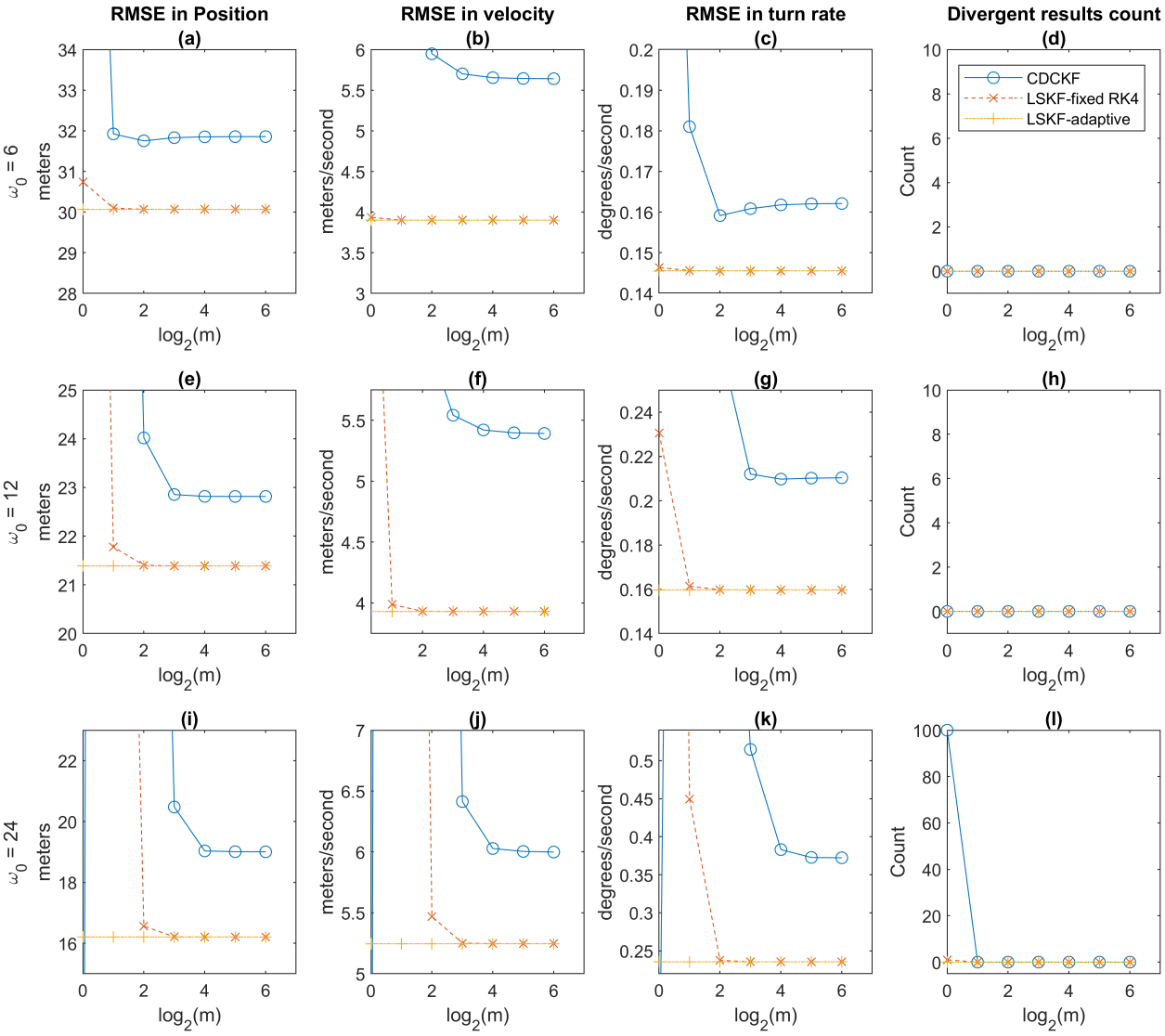


Figure II.3: RMSE and count of divergence results, for a fixed measurement interval $T = 6s$ varying m and ω_0, where m is the number of timestep subdivisions between measurements, and ω_0 is the initial turn rate. Each row gives the performance metrics for the same initial turn rate, whereas each column contrasts the same performance measurement across different initual turn rates. Note for $\omega_0 = 24°/s$ and $m = 1$, all results from CD-CKF are divergent.

Our numerical results in Fig. II.3 show that our methods consistently outperform the CD-CKF in this test case, across all choices of angular velocity and timestep subdivisions.

Importantly, note that CD-CKF performance cannot match the LSKF-adaptive even if sufficient timestep subdivisions are introduced. We suspect this is due to the fact that the CD-CKF as introduced in [AHH10] only uses the IT-1.5 expansion at the beginning of each *time-update* step but not at the subdivided timesteps, whereas our method is defined using instantaneous information and is not subject to this limitation.

Equally importantly, note that the LSKF-adaptive version of our method gives the same result independent of the subdivisions introduced. Additionally, the fixed-timestep LSKF-RK4 converges to the LSKF-adaptive result, as expected for a consistent ODE solver. In practice, a user can always use the LSKF-adaptive version with the choice of adaptive ODE solvers that gives the best performance without needing to consider timestep subdivisions manually.

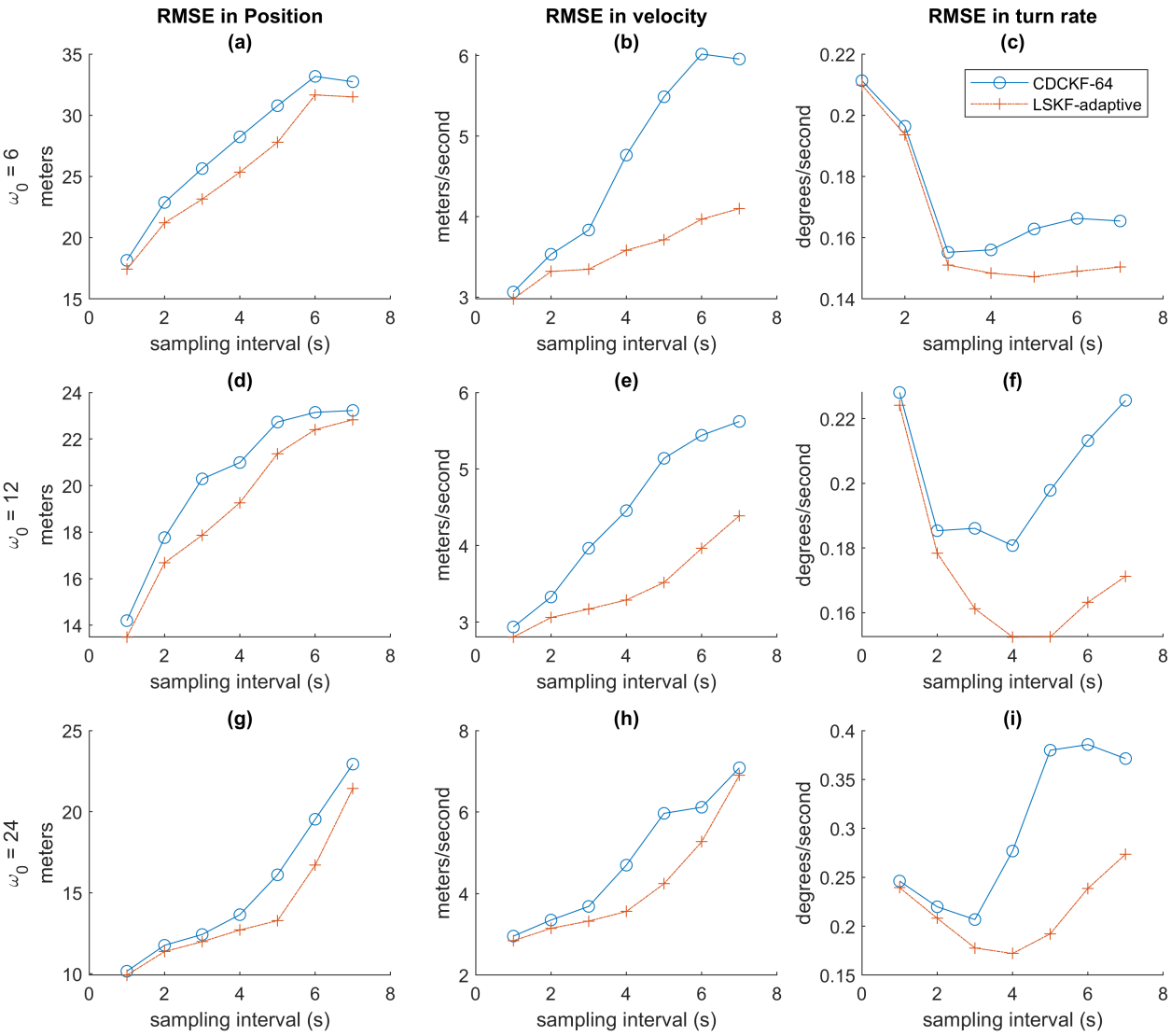


Figure II.4: RMSE with varying the measurement interval from 1 through 7 seconds, and varying the initial turn rate ω_0. For the CD-CKF, a timestep subdivision of $m = 64$ is chosen to ensure that the CD-CKF is performing optimally. For the LSKF, an adaptive ODE solver is used and no timestep subdivisions are manually inserted. All results from CD-CKF and LSKF are convergent.

In Fig. II.4, we verify that even with sufficient timestep subdivision for the CD-CKF, the LSKF still outperforms CD-CKF over all the parameters chosen, even when no intermediate

timesteps are manually inserted. Additionally, the difference in performance between CD-CKF and LSKF is more significant when the measurement interval T is large, whereas the results are similar when $T = 1s$. With this in mind, we proceed further into the numerical experiments, using sufficient timestep subdivisions ($m = 64$) for the CD-CKF, whereas no additional pre-defined timestep subdivisions ($m = 1$) for the LSKF-adaptive, and vary the measurement interval T from 1 second to 7 seconds with an increment of 1 second.

In conclusion, for the test case picked by [AHH10], our LSKF method consistently outperforms CD-CKF across the challenging scenarios introduced by them. In addition, our method requires less input from an end-user, as our method only requires knowledge of the *drift function* explicitly, whereas CD-CKF also requires the first and second spatial derivatives of the *drift function*, as well as a user-defined timestep subdivision parameter m. Finally, the elegance of reforming the system as an ODE without introducing expansion in time gives more room for possible future improvement.

II.4.3: Tolerance to initial guess

In the previous numerical example, we set the initial guess $\mathbf{x}_0$ to be exact and overestimated the initial covariance matrix, which is a common practice, and the initial estimation is usually expected to have a negligible effect on the accuracy of the result with many measurements taken [LSRJ11]. However, for this challenging scenario with a 7-dimensional state space but a 3 dimensional measurement, it turns out that the choice of initial covariance also has an impact on the performance of the filtering algorithms. Here, we compare the performance of the LSKF and CD-CKF with: a) a *best guess* where the initial covariance is approximately equal to the RMSE, and b) a *bad guess* where the initial covariance underestimates the error in the initial guess.

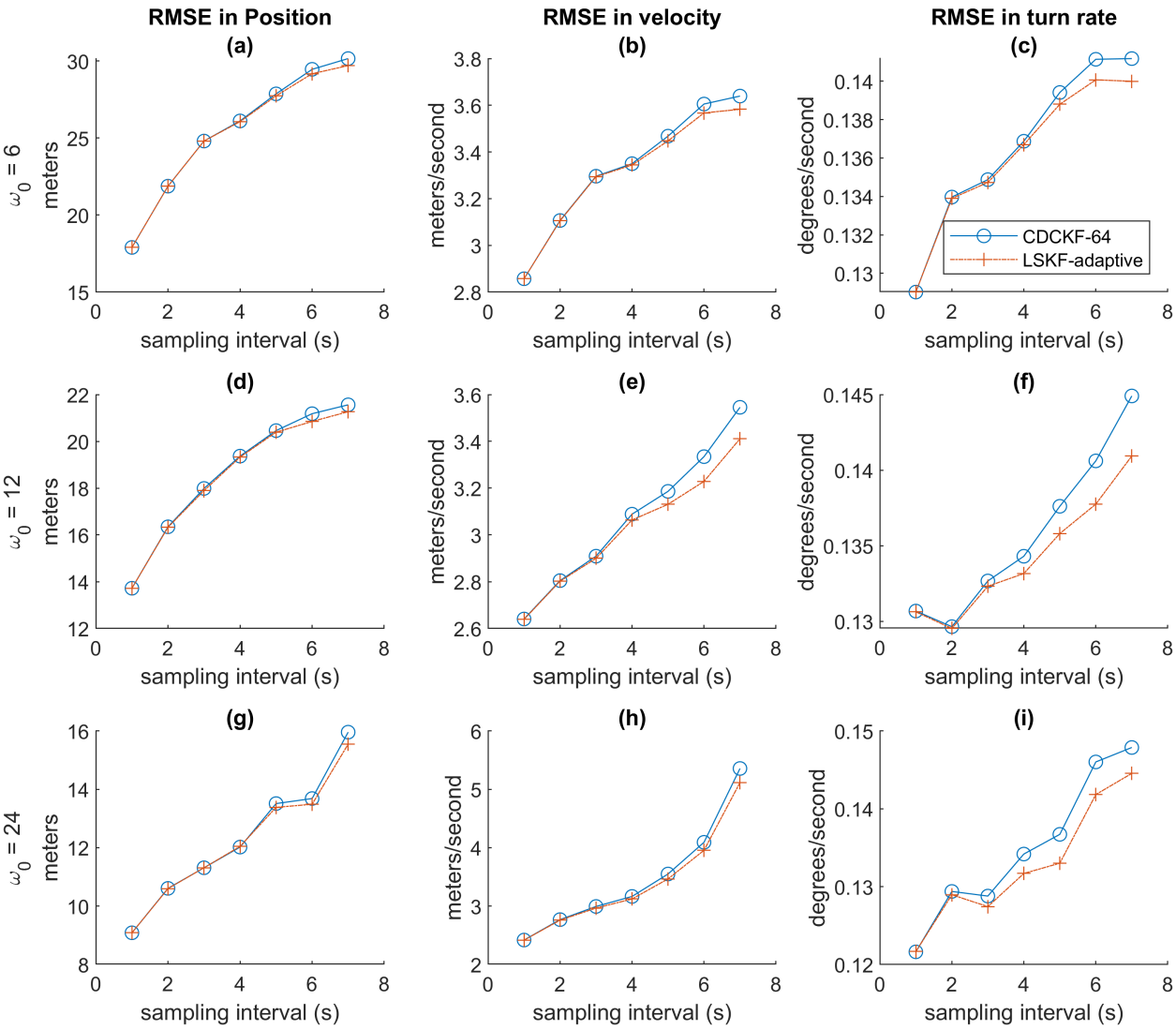


Figure II.5: RMSE with varying the measurement interval from 1 through 7 seconds, and varying the initial turn rate ω_0. The setting is the same as Fig. II.4 except that the guess for standard deviation of the turn rate is reduced from 0.1 to 0.00349. All results from CD-CKF and LSKF are convergent.

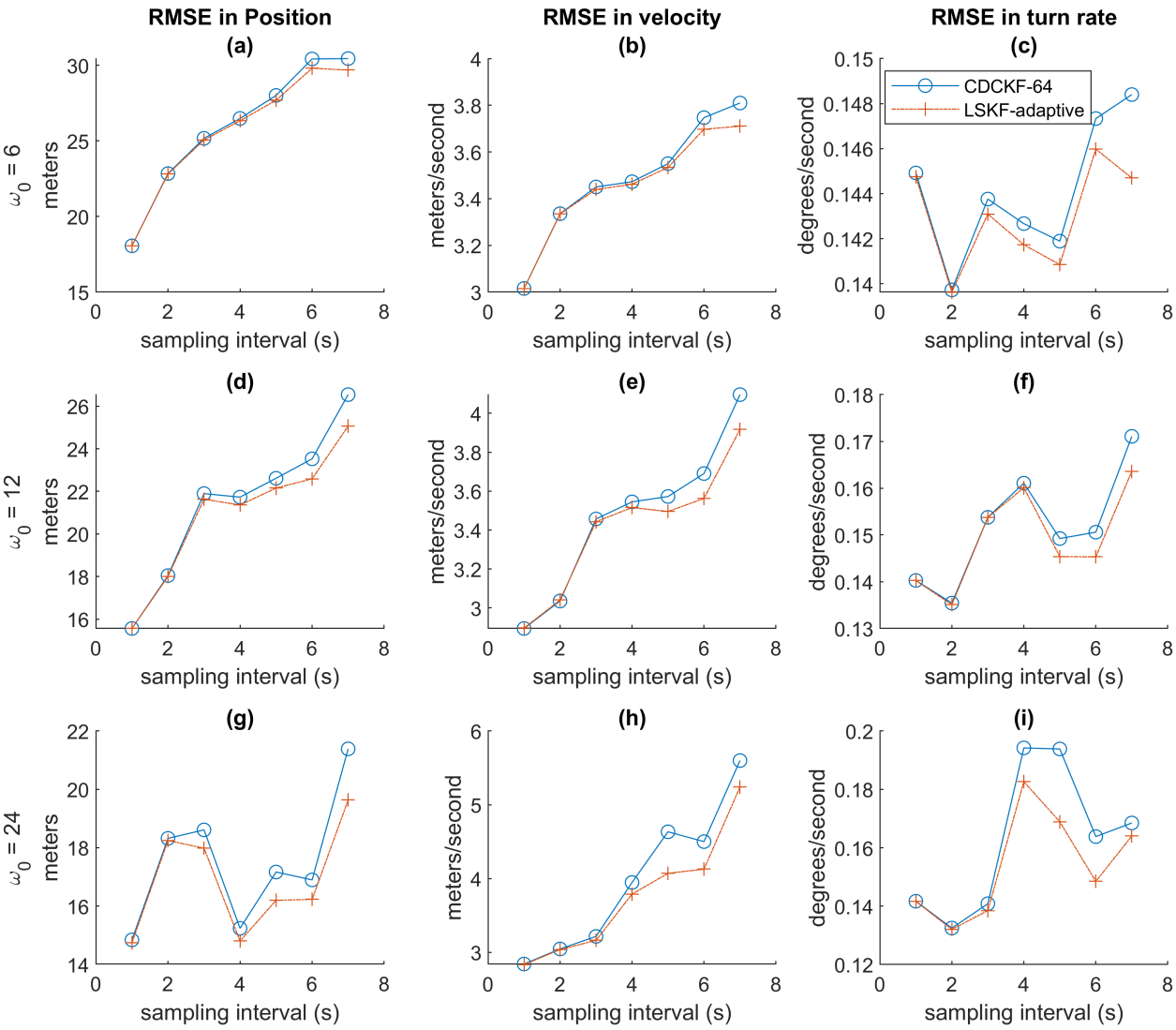


Figure II.6: RMSE with varying the measurement interval from 1 through 7 seconds, and varying the initial turn rate ω_0. The setting is the same as Fig. II.5 except that the a zero mean random error with covariance $\mathbf{\Sigma}_0$ is added, but the initial guess used for estimation is underestimated to be $\frac{1}{16}\mathbf{\Sigma}_0$. All results from CD-CKF and LSKF are convergent.

In the *best guess* scenario, we keep the settings the same as presented in Sec. II.4.2, except that the standard deviation of the turn rate is reduced from 0.1 radian to 0.00349 radian, or approximately 0.2 degree. Comparing Fig. II.4 and Fig. II.5, we conclude that the LSKF still outperforms the CD-CKF if the initial guess of covariance is close to the average RMSE.

Additionally, the LSKF is less impacted by an initial covariance overestimation than the CD-CKF. This also explains that the poor performance for both filtering methods in predicting the turn rate, as shown in Fig. II.4, is caused by the overestimated variance in turn rate. Since the dynamics are highly nonlinear in the turn rate, and the limited information on the initial turn rate impacts the performance of the filter even with frequent measurements.

In the *bad guess* scenario, we further add an error to the initial guess, generated by a zero-mean normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_0)$ as defined in the *best guess* scenario, but feed the system with an initial guess for the covariance matrix $\frac{1}{16}\mathbf{\Sigma}_0$, which underestimates the error in the initial condition. Comparing Fig. II.6 with Fig. II.5, we note that the performance advantage of LSKF over CD-CKF is larger with an *underestimated* initial covariance.

II.5: Conclusion

Herein, we derived a novel Level Set Kalman Filter method for nonlinear continuous-discrete systems. From a theory standpoint, our derivation is based on the movement of a level set instead of the moments of a distribution. Our method reformulates the *time-update* of the filtering as an ODE. As a consequence of this formulation, our description is instantaneous, in contrast to existing methods that use some expansion in time to approximate the continuous process noise.

From a practical point, for the radar tracking coordinated turn test case, our method consistently outperforms the CD-CKF over a range of challenging scenarios. Additionally, our method requires less explicit information about the model, and our instantaneous formulation allows a user to easily pass the task of choosing a timestep to the well-established field of adaptive ODE solvers. It is also more tolerant to varying initial guesses. The numerical results indicate that our method is a good candidate for challenging tracking problems, especially if an appropriate timestep cannot be determined *a priori*.

Appendix A: Comparison of the standard, averaged and partially averaged time-update of the LSKF

In equations (II.3.28) and (II.3.34), we defined the **(standard)** *time-update* and the **averaged velocity** *time-update* equations. Here we use a numerical example to illustrate the difference in behavior between these methods. Additionally, we introduce the **partially averaged velocity** *time-update* equations as a trade-off option between the standard and the averaged velocity version.

Using the same notations as in (II.3.28) and (II.3.34), the **partially averaged velocity** is defined as

$$\mathbf{v_p} := \frac{1}{2d}\left(\sum_{i=1}^{d}(\mathbf{v}(\mathbf{x}+\mathbf{x}_i)) + d \times \mathbf{v}(\mathbf{x}-\mathbf{x}_1)\right). \tag{II.5.1}$$

Correspondingly, in matrix form, the *time-update* ODE using partially averaged velocity is defined by

$$\frac{\partial \mathbf{M}}{\partial t} = \mathbf{v}(\mathbf{x}+\mathbf{M}) - \mathbf{v}_p + \frac{1}{2}\mathbf{K}(\mathbf{M}^T)^{-1}. \tag{II.5.2}$$

As an illustrative example to show the effects of using an *averaged velocity* or *partially averaged velocity* versus evaluating the velocity at the center when a nonlinear drift velocity is present, we consider the following system:

$$\frac{\partial u}{\partial t} = -\nabla \cdot (\mathbf{v}u),$$

where a, b are positive constants, and

$$\begin{aligned}\mathbf{v}(x,y,0) &= (0, x^2)\\ u(x,y,0) &= \frac{1}{2\pi ab}\exp(-\frac{\frac{x^2}{a^2}+\frac{y^2}{b^2}}{2}).\end{aligned}$$

(Note x, y in the following equation are not in bold, and are scalars) The analytic solution to this transport equation is given by

$$u(x, y, 0) = \frac{1}{2\pi ab} \exp(-\frac{\frac{x^2}{a^2} + \frac{(y-xt)^2}{b^2}}{2}).$$

To compare the performance of the three LSKF methods, we compute the averaged $\mathbf{L^2}$ error of the results with the analytical result, with randomly chosen matrix square root $\mathbf{M}\mathbf{M}^T = \mathbf{\Sigma}$. As can be observed from Fig. II.7, the partially averaged velocity and averaged velocity version has less RMSE than the standard method. The averaged velocity method is less sensitive than the partially averaged method in that the covariance entries are less dependent on the choice of matrix square root. However, since the partially averaged method requires $d+1$ evaluations of the drift velocity whereas averaged method requires $2d$ evaluations, the partially averaged method is still useful as it can be considered as an efficient improvement from the standard version.

Appendix B: Computational cost of the LSKF

A count of FLOPs would be misleading for this method, as the main function reformulates the function as an ordinary differential equation, and the number of steps used is highly dependent on the choice of the numerical ODE solver and the numerical properties of the problem when an adaptive solver is used. When using an ODE solver, most of the computational cost is associated with evaluating the derivative. Therefore, we find the number of evaluation of the **drift velocity v**, and FLOPs needed for a single derivative evaluation in (II.3.34). The cost for computation for the *time-update* is then mainly decided by the number of derivative evaluations needed and the length of the timestep. The *measurement-update* is identical to that of CD-CKF, which is listed in Table V of [AHH10] and is omitted here.

Computations needed for (II.3.34):

1. Evaluate **averaged velocity**:

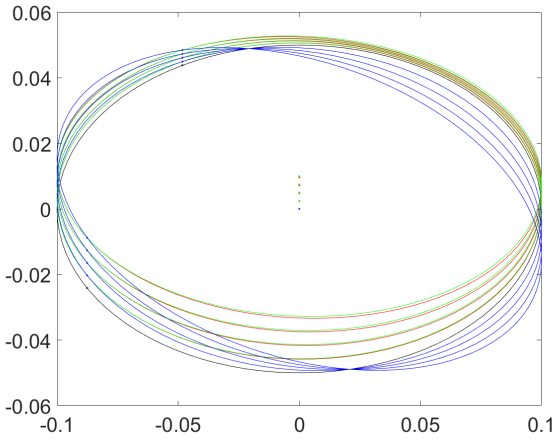
$$\frac{\partial \bar{\mathbf{x}}}{\partial t} = \mathbf{v}_a(\bar{\mathbf{x}}, \mathbf{M}) := \frac{1}{2d} \sum_{i=1}^{d} (\mathbf{v}(\bar{\mathbf{x}} + \mathbf{x}_i) + \mathbf{v}(\bar{\mathbf{x}} - \mathbf{x}_i)). \tag{II.5.3}$$

   $2d$ **drift velocity** evaluations and $O(d^2)$ FLOPs.

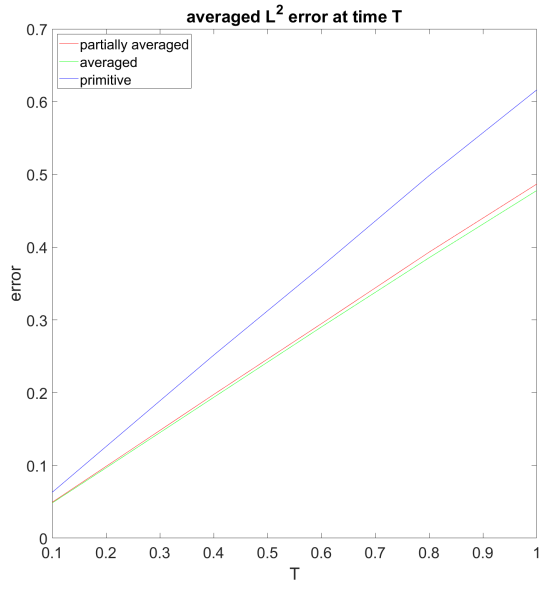
2. Find $\mathbf{K}(\mathbf{M}^T)^{-1}$: if solved by LU factorization, forward substitution, and back substitution: $\frac{8}{3}d^3 + O(d^2)$ FLOPs.

3. Evaluate (II.3.34): $O(d^2)$ FLOPs.

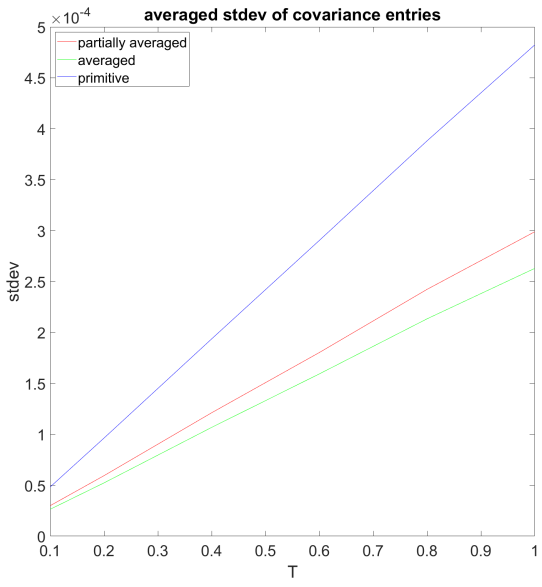
In total, $2d$ **drift velocity** evaluations and $\frac{8}{3}d^3 + O(d^2)$ FLOPs are needed for evaluating (II.3.34). When using a fixed Runge-Kutta 4 method, 4 such evaluations are needed per timestep, which results in $8d$ **drift velocity** evaluations and $\frac{32}{3}d^3 + O(d^2)$ FLOPs per *time-update*. If the measurement interval is short, then a fixed Runge-Kutta 2 method can be used, with half derivative evaluations needed. For most scenarios, we suggest using an adaptive solver to automatically determine the appropriate timestep given a target error bound.



A



B



C

Figure II.7: comparison between the partially averaged (red), averaged (green), and standard (blue) LSKF applied to a nonlinear system. **A** shows the trajectories of the center and level set. **B** shows the averaged $\mathbf{L^2}$-norm of error over 1024 trials. **C** shows the averaged standard deviation of covariance entries over those trials. Quantities in this figure are unitless.

CHAPTER III

The Asymmetric Particle Population Density Method

III.1: Introduction

Many biological behaviors can be modeled by coupled oscillating elements. Examples on various scales include the rhythmic flashing of fireflies [Buc88], neurons [HHK52], and circadian rhythms [THHDI07]. Understanding the macroscopic behavior (e.g., synchronization of the population) for a large population without calculating the dynamics of all oscillators individually is of great value. A common approach to this problem is to consider the population density distribution [Win67, JR95, San96, NT00, Kur03]. However, most existing methods also have severe constraints on the oscillator model, such as requiring the oscillator to closely follow a limit cycle or that the coupling is weak. These constraints and simplifications can leave some important applications behind, such as performing a parameter search to tune the parameters that generate desired macroscopic properties. Additionally, since noise can also generate macroscopic patterns [GSB11, KYW+10], it needs to be accounted for accurately. The goal of our proposed method is to form a middle ground between the direct simulation of all oscillators and existing population density-based methods, such that: 1) the full dynamics of the oscillator are retained; 2) the method accurately accounts for the noise; 3) the method accommodates complicated coupling between the oscillators, and; 4) the method is faster than the direct simulation.

Here, we present the asymmetrical particle population density (APPD) method to sim-

ulate such a population of noisy all-to-all-coupled oscillators. Our method relies on a population density approach, which tracks the probability that any element is in any particular state at a time. While such methods are widely considered computationally prohibitive for biologically realistic models, Stinchcombe and Forger [SF16] note that most of the models are dissipative. Therefore, only a fraction of the possible system states have a non-negligible number of oscillators at any given time. Additionally, coupling between the oscillators can concentrate the distribution. As demonstrated in [SF16], these distributions can be simulated using particle methods without any further mathematical simplifications. However, the particle method presented in [SF16] is still limited by two factors: 1) Particles are symmetric in space, which covers a trajectory inefficiently. 2) The computation of the noise term is based on the interaction between particles [MG02], thus requiring more computation and being more difficult to parallel than a method where particles can be updated independently. We address these issues to create a fast and efficient method.

While the APPD method is inspired by the previous work [SF16] using a particle method in a similar setting, it is novel in that the particles are asymmetric, allowing them to change their shape with the flow. This allows the population density to be very narrow in some dimensions. In practice, we find this true in neuronal simulations, which greatly reduces the number of particles needed and increases these methods' accuracy. Additionally, the particles can be updated individually except for coupling, allowing a much easier and more efficient parallel implementation of the algorithm.

The practice of using an asymmetric deforming kernel to solve a convection-diffusion equation or a Fokker-Planck equation is not new, either in the context of convection-diffusion equation [Ros96, Ros05], or in a very different context of Kalman-Bucy filtering [Sar07, AHH10, GH11, KK17, WF21b]. The concept of splitting and combining such particles is also known [XPP+14, ABD14, BBH21]. But our main innovation is as follows: First, the derivation of our method is based on tracking movement of a level set, which is very different from existing methods based on a direct computation of terms in the covariance

matrix. As a result, our method naturally tracks a *square-root* of the covariance, which gives improved numerical stability, as known in the Kalman filter community [Sar07, AHH10] but has not been introduced to this context. This improved stability is needed as previous particle methods are restricted to 2-dimensional scenarios, whereas the Hodgkin-Huxley model [HHK52] has a 4-dimensional state space. Additionally, our method does not require a Taylor expansion in time with an *a priori* timestep to represent diffusion, which enables an adaptive timestep method to be directly applied. Also, the splitting of a single particle into three particles with optimized weight and distance introduces less error than a 2-particle split, as in [BBH21], for example. For the sake of simplicity, in the following discussion, we will refer to elements as oscillators, a key target application. However, this method would also apply to systems that are not strictly oscillators, such as if they are quasiperiodic.

III.2: Method

III.2.1: Overview of the method:

For simulating a large population of coupled noisy oscillators, we consider the direct Monte-Carlo simulation as the baseline method that we compare against. In the direct Monte-Carlo method, each oscillator is updated individually based on its dynamics, noise, and coupling. In our proposed APPD, the intuition for the method is simple: a particle not only represents oscillators exactly at this state but also nearby oscillators with some locally linear approximation. In the following parts, we present how such intuition is implemented, namely: how the dynamics and noise of the system are updated for a single particle in a deterministic algorithm, how the local linearity is preserved by splitting the particle, and other technicalities that arise.

In the APPD, a single particle is a weighted multivariate Gaussian. Simulating the probability distribution function (PDF) of a similar stochastic process using a single multivariate Gaussian is a well-studied topic, in the context of Kalman-Bucy filtering [KB61] and its gen-

eralizations [Sar07, AHH10, GH11, KK17]. These methods utilize the fact that a Gaussian function is preserved if the dynamics of the system are linear in space. Therefore, when the PDF is concentrated in a small region, it can be considered approximately locally linear, and the Gaussian is preserved. However, these existing methods [Sar07, AHH10, GH11, KK17] are based on Itô-Taylor expansion of the underlying equation; thus, the representation of noise is dependent on the choice of the timestep size. In Cha. II, we proposed the Level Set Kalman Filter (LSKF), whose representation is independent of the time discretization and has superior accuracy than the Continuous-Discrete Cubature Kalman Filter. Therefore we base the particle in our purposed method on the *time-update* of the LSKF.

While the Fokker-Planck equation for the LSKF and the convection-diffusion equation for the population takes the same form, the challenges for porting the LSKF or similar multivariate-Gaussian-based methods to the context of coupled noisy oscillators are non-trivial. First, oscillators can have a limit cycle with strong contraction, resulting in the PDF being very thin in certain directions, corresponding to a close-to-singular covariance matrix in a multivariate-Gaussian approximation. Therefore the method has to be robust with respect to singular covariance. For the LSKF, since it works with a square root of the covariance matrix and is robust for semi-positive definite covariances, it is stable as long as a backward-stable linear equation solver is used and the near-singular eigenvectors are dropped. The next challenge is that the approximately linear condition for the dynamics is not valid for oscillators; additionally, we are interested in more complicated distributions than what can be described by a single multivariate Gaussian. This challenge is resolved by introducing a scheme to split and combine particles.

III.2.2: Problem formulation

The subject of the simulation is a population consisting of identical oscillators subject to dynamics described by a velocity field $\mathbf{v}$, and a noise that can be described by a Wiener process. We assume that an oscillator can be described by a d-dimensional state variable.

Suppose that the oscillators are also subject to noise, then this probability density u is evolved according to the following convection-diffusion equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u), \tag{III.2.1}$$

in which $\mathbf{K}$ is a constant **diffusion matrix**, and $\mathbf{v} = \mathbf{v}(\mathbf{x}, u)$ is the **convection velocity** that corresponds to the ordinary differential equation of the oscillator, and the coupling effect between oscillators.

We further assume that the population is large (in fact, infinitely large) such that individual fluctuations will not affect the population. Hence the probability distribution and the population distribution are the same. This assumption is essential as the coupling would be more complex otherwise.

To accommodate coupling, we assume the velocity field takes the following form:

$$\mathbf{v} = \mathbf{v}_d(\mathbf{x}) + \mathbf{v}_c(u, \mathbf{x}), \tag{III.2.2}$$

in which $\mathbf{v}_d$ is the **dynamic velocity** corresponding to the dynamics that are determined by the state variable $\mathbf{x}$ only. $\mathbf{v}_c$ is the **coupling velocity** corresponding to the coupling term that depends on the state variable $\mathbf{x}$, and the population distribution u at the same time. Moreover, we assume the coupling velocity $\mathbf{v}_c$ is of the following form:

$$\mathbf{v}_c(u, \mathbf{x}) = \mathbf{w}_c(\mathbf{x}) \cdot \mathbf{L}[u], \tag{III.2.3}$$

in which $\mathbf{w}_c(\cdot)$ is a vector-valued function, $L[\cdot]$ is a linear functional on the PDF u.

In principle, the dependence of coupling velocity on the density makes (III.2.1) a nonlinear differential equation, which gives rise to complex behaviors. However, since the coupling velocity $\mathbf{v}_c$ as defined in (III.2.3) depends on global quantities of u over the domain and can be assumed to change rather slowly for our applications. Therefore, we approximate

$\mathbf{v}_c = \mathbf{v}_c(\mathbf{x})$ to be independent of u during the update, except when the velocity function $\mathbf{v}_c$ is updated at some fixed timesteps. This approximation keeps (III.2.1) linear in the following derivations.

To numerically simulate the population density, a discretization is required. In a **particle method**, the population density is discretized by approximating the population density as a linear combination of **particles**:

$$u(\mathbf{x}) \approx \sum_{i \in I} w_i K_i(\mathbf{x} - \mathbf{x_i}), \tag{III.2.4}$$

where for each particle with index i, $K_i(\mathbf{x})$ is the **kernel function** of the particle, w_i is the **weight** of the particle, and $\mathbf{x_i}$ is the **center location** of the particle.

In our method, each particle is a weighted multivariate Gaussian. In particular, each Gaussian particle is represented by its **weight** w_i, **center location** $\mathbf{x}_i$ and **covariance matrix** $\mathbf{\Sigma}_i$, and its **kernel function** K_i is given by:

$$K_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma}_i)}} \exp\left(-\frac{1}{2}\mathbf{x}^T \mathbf{\Sigma}_i^{-1} \mathbf{x}\right). \tag{III.2.5}$$

To improve the numerical stability and derive a simpler update algorithm, we track and store a **square root** of the covariance matrix $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$ instead, and only evaluate the covariance matrix when needed.

III.2.3: Single particle update

This section describes the particle update algorithm, which is a special case of the Level Set Kalman Filter method as derived in Cha. II. Only the description is included here. Therefore readers interested in the derivation and proof should refer to Cha. II.

Recall (III.2.1):

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u). \tag{III.2.6}$$

For a single particle centered at $\mathbf{x}_0$, the probability density function is given by

$$u(\mathbf{x}, 0) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \exp\left(-\frac{(\mathbf{x}-\mathbf{x}_0)^T \mathbf{\Sigma}^{-1} (\mathbf{x}-\mathbf{x}_0)}{2}\right). \tag{III.2.7}$$

Consider the **level set** of the function F:

$$\left\{\mathbf{x} | F(\mathbf{x}, t) := \frac{u(\mathbf{x}, t)}{u(\mathbf{0}, t)} = c\right\} \quad (0 < c < 1). \tag{III.2.8}$$

For u being a Gaussian particle, all level sets are ellipsoids. Tracking the movement of the Gaussian particle is equivalent to tracking one of its ellipsoid level sets as defined in (III.2.8). A factorization of the covariance matrix $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$ (where $\mathbf{M}$ is called a **square root** of $\mathbf{\Sigma}$) represents a level set in the sense that the column vectors lie on the same ellipsoid. We represent the columns of the matrix $\mathbf{M}$ as $\mathbf{M}_i$, and advance them in time. The ordinary differential equation is given by:

$$\frac{\partial \mathbf{M}_i}{\partial t} = \frac{1}{2}\left(\mathbf{v}(\mathbf{x}_0 + \mathbf{M}_i) - \mathbf{v}(\mathbf{x}_0 - \mathbf{M}_i)\right) + \mathbf{K}(\mathbf{M}^T)^{-1}\mathbf{e}_i, \tag{III.2.9}$$

where $\mathbf{x}_0$ is the **center location** of the Gaussian particle, and the matrix inverse shall be understood as solve the equation with a backward-stable solver.

A matrix short-hand form is as follows (where the matrix-vector additions are defined entrywise):

$$\frac{\partial \mathbf{M}}{\partial t} = \frac{1}{2}\left(\mathbf{v}(\mathbf{x}_0 + \mathbf{M}) - \mathbf{v}(\mathbf{x}_0 - \mathbf{M})\right) + \mathbf{K}(\mathbf{M}^T)^{-1}, \tag{III.2.10}$$

and the velocity for center is given by:

$$\frac{\partial \mathbf{x}_0}{\partial t} = \frac{1}{2d}\sum_{i=1}^{d} \mathbf{v}(\mathbf{x}_0 + \mathbf{M}_i) + \mathbf{v}(\mathbf{x}_0 - \mathbf{M}_i). \tag{III.2.11}$$

Notice that to evaluate the velocity for one point $\mathbf{x}_i$ on the level set, the center location, and

all other points $\mathbf{x}_j$ for this Gaussian kernel are needed, hence the points on a level set cannot be updated independently, and the dimension for the ODE solver is $d \times (d+1)$. However, the updates of different Gaussian particles are independent.

As a summary, a particle is updated as follows:

Algorithm 2 summary of particle update

Require: center $\mathbf{x}$, and a square root $\mathbf{M}$ of the covariance matrix at previous time step.

Pass $\mathbf{M}$ and $\mathbf{x}$ as the state variable to an ODE solver with derivative defined by (III.2.10).

return center $\mathbf{x}'$, and a square root $\mathbf{M}'$ of the covariance matrix.

III.2.4: Splitting a particle

Since updating the center location and the square root of the covariance matrix for a single particle uses a linear approximation of the convection velocity, we need a method to control the size of the Gaussian particle that would otherwise expand due to diffusion. Though the Gaussian particles have infinite support, most of the density is within 2 standard deviations in each eigenvector direction, and we consider this region to be its effective support. The criterion we use for checking the linear approximation is as follows: we check the soundness of the linear approximation in the direction of the off-center points used in the particle update. We decide if the particle needs to be split if this particular relative error for any of the off-center points is larger than a threshold chosen by the user:

$$\epsilon = \frac{\|(\mathbf{v}(\mathbf{x}+2\Delta\mathbf{x}) - \mathbf{v}(\mathbf{x})) - 2(\mathbf{v}(\mathbf{x}+\Delta\mathbf{x} - \mathbf{v}(\mathbf{x})))\|}{2\|\mathbf{v}(\mathbf{x})\|}, \tag{III.2.12}$$

where $\mathbf{x}$ is the center of the particle, and $\Delta\mathbf{x}$ is the offset of the off-center point. For the numerical examples that follow, the value is chosen as 0.05.

Once the velocity field v in the effective support deviates from the linear approximation larger than this tolerance, we need a method to reduce the covariance, thereby reducing the effective support of the particle. We achieve this by splitting the particle into three thinner particles in the corresponding direction with half the variance in that direction.

Up to some rotation, the kernel function (III.2.5) can be reformulated as:

$$K_i(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\sigma_1\sigma_2\dots\sigma_d}} \exp\left(-\frac{1}{2}\left(\frac{x_1^2}{\sigma_1} + \frac{x_2^2}{\sigma_2} + \cdots + \frac{x_d^2}{\sigma_d}\right)\right), \tag{III.2.13}$$

in which $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$ are the eigenvalues of $\mathbf{\Sigma}$. We here consider a method to reduce the width of the population in the x_1 direction by representing the original population as a summation of three particles with variance in the x_1 direction reduced by half. The particle in the center would weigh $1 - 2\omega$, and have its location the same as what it replaced, while the two off-center particles would both weigh ω, and have its center location shift by a and $-a$ in the direction of x_1 respectively. Note the weight of the three children particles add up to 1, and the total weight of the particle is conserved after the operation.

We would like to minimize the error introduced measured in the infinity norm, subject to the constraint that the total weight is constant. This reduces this problem to a simple minimization that is independent of the other dimensions of the state space:

$$\min_{a,\omega} \max_{x} |K_{2d}(x) - ((1-2\omega)K_d(x) + \omega K_d(x-a) + \omega K_d(x+a))| \tag{III.2.14}$$

Where $K_d(x)$ is the probability density function of the normal distribution with variance d. Using numerical optimization, we find $a = 1.03332\sqrt{d_1}$, and $\omega = 0.21921$. The process is then repeated to other directions, until variance in all directions are sufficiently small. This process introduces a maximum relative error of 0.73%. This is enough to bias a careful analysis of convergence as the size of particles is reduced (and thus, we do not provide convergence results). However, it does offer sufficient accuracy in our simulations since the error introduced has little effect on the linear coupling velocity term. Therefore, the error introduced does not significantly affect the global behavior of the population, as will be shown in the examples below. If higher accuracy is required, one needs to reduce the difference between the variance of the particles. A demonstration of the split is shown in Fig. III.1.

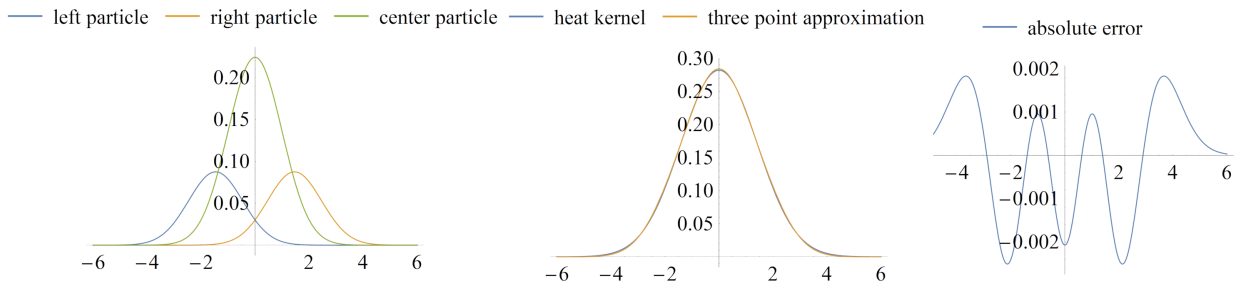


Figure III.1: Optimal particle split along an eigen-direction. The left figure plots 3 components of the split particle. The yellow curve is the new particle in the center, while the green and red curve are the shifted particles. All three have half the variance as the density they approximate. The middle figure gives a comparison between a normal distribution (blue) and its approximation (yellow) with 3 distributions with half the variance. The right figure shows the absolute error introduced.

Algorithm 3 Particle Split (Eigen-decomposition)

Require: Particle with weight 1 centered at origin with a Diagonal covariance matrix $\mathbf{\Sigma} = \text{diag}([\sigma_1, \ldots, \sigma_d])$

Define the new Diagonal covariance matrix $\mathbf{\Sigma}' = \text{diag}([\sigma_1/2, \sigma_2, \ldots, \sigma_d])$

Find center locations of the off-center particles given by $\mathbf{x}_c = \left[1.03332\sqrt{\sigma_1}, 0, \ldots, 0\right]^T$

Construct center particle centered at origin, with weight $1 - 2\omega$ and covariance $\mathbf{\Sigma}'$

Construct left and right particle centered at $\pm\mathbf{x}_c$, with weight ω and covariance $\mathbf{\Sigma}'$

Apply this method to the new particles until all variance are sufficiently small

The above derivation uses the eigendecomposition of the covariance matrix, which could be expensive and undesirable in some applications. While the choice of a and w may not be necessarily optimal for the split, the particle split can be applied to any (non-eigen) direction similarly by substracting appropriate orthogonal projections detailed as follows:

Given a particle centered at origin with weight 1, and a covariance matrix given by its

square root decomposition $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$, suppose split is needed along direction $\mathbf{M}_1$, then the children particles are computed as follows:

- Center particle: centered at the origin, weight $1 - 2\omega$, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$
- Left particle: centered at $1.03332\mathbf{M}_1$, weight ω, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$
- Right particle: centered at $-1.03332\mathbf{M}_1$, weight ω, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$

in which $w = 0.21921$, and the matrix $\mathbf{N}$ defined column-wise by:

$$\mathbf{N}_i = \mathbf{M}_i - \left(1 - \frac{1}{\sqrt{2}}\right) \frac{< \mathbf{M}_1, \mathbf{M}_i >}{< \mathbf{M}_1, \mathbf{M}_1 >} \mathbf{M}_1 \tag{III.2.15}$$

in which $< \cdot, \cdot >$ denotes the inner product.

Algorithm 4 Particle Split (square root)

Require: A particle with weight 1 centered at origin and a square root decomposition $\mathbf{M}\mathbf{M}^T$ of the Covariance matrix

Find new square root decomposition $\mathbf{N}$ define in (III.2.15)

Find center locations of the off-center particles $\mathbf{x}_c = 1.03332\mathbf{M}_1$

Construct center particle centered at origin, with weight $1 - 2\omega$ and decomposition $\mathbf{N}$

Construct left and right particle centered at $\pm\mathbf{x}_c$, with weight ω and decomposition $\mathbf{N}$

Apply this method to the new particles until all variance are sufficiently small

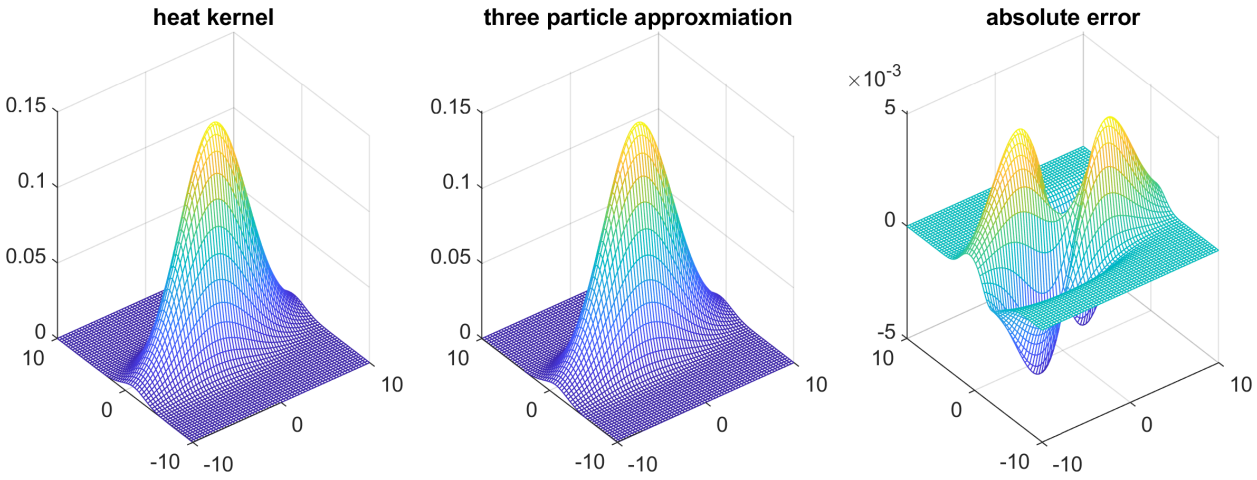


Figure III.2: Particle split along arbitrary direction.The left figures shows the PDF of a particle with covariance $\begin{bmatrix} 16 & 2 \\ 2 & 3 \end{bmatrix}$. The middle figure and right figure shows the three particle approximation, and absolute error introduced when the particle is split along $[0, 1]^T$ direction, which is not an eigenvector.

It can be seen in Fig. III.2 that while the absolute error introduced is larger than along eigen-direction, the split particles still closely approximate the particle they replaced. This split works with the square root representation and avoids the costly eigendecomposition.

III.2.5: Combining particles

The previous method introduces new particles to the system. Therefore, a method to remove particles is required since otherwise, the number of particles can grow geometrically, which would render this method impossible to use.

The description of the method for combining the particles can be divided into two parts: first, given a set of particles, a method to combine these particles to a single one optimally; second, a strategy to determine which particles should be merged into a single one.

First, consider the method for combining particles. Except that the total weight does not add to 1, we are finding a multivariate normal distribution that best approximates the

PDF of the old population. Therefore we choose the new particle such that it has the same mean (center location) and covariance matrix of the distribution, and the weight to be the sum of all the particles to be combined, conserving the total weight.

Consider a probability distribution with PDF, note in the following part: $\boldsymbol{\mu}_i$ is the expectation vector of ith particle, whereas μ_i is the ith component of a vector $\boldsymbol{\mu}$:

$$p(\mathbf{x}) = \sum_i \frac{w_i \sqrt{\det(\boldsymbol{\Sigma}_i)}}{(4\pi)^{d/2}} \exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)}{2}\right). \tag{III.2.16}$$

Then the expectation of the new distribution μ is the weighted average of μ_n:

$$\boldsymbol{\mu} = \frac{\sum_n w_n \boldsymbol{\mu}_n}{\sum_n w_n}. \tag{III.2.17}$$

The ijth entry of the covariance matrix $\boldsymbol{\Sigma}$ is given by

$$\boldsymbol{\Sigma}_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)] = \sum_n \frac{w_n}{\sum_m w_m} E[(Y_{ni} - \mu_i)(Y_{nj} - \mu_j)], \tag{III.2.18}$$

where Y_n is a multivariate normal random variable subject with expectation $\boldsymbol{\mu}_n$ and covariance matrix $\boldsymbol{\Sigma}_n$. Hence the expectation in III.2.18 can be evaluated as:

$$\begin{aligned} E[(Y_{ni} - \mu_i)(Y_{nj} - \mu_j)] &= E[(Y_{ni} - \mu_{ni} + (\mu_{ni} - \mu_i))(Y_{nj} - \mu_{nj} + (\mu_{nj} - \mu_j))] && \text{(III.2.19)} \\ &= E[(Y_{ni} - \mu_{ni})(Y_{nj} - \mu_{nj})] + (\mu_{ni} - \mu_i) E(Y_{nj} - \mu_{nj}) \\ &\quad + (\mu_{nj} - \mu_j) E(Y_{ni} - \mu_{ni}) + (\mu_{ni} - \mu_i)(\mu_{nj} - \mu_j) && \text{(III.2.20)} \\ &= (\boldsymbol{\Sigma}_n)_{ij} + (\mu_{ni} - \mu_i)(\mu_{nj} - \mu_j), && \text{(III.2.21)} \end{aligned}$$

in which μ_{nj} is the jth component of $\boldsymbol{\mu}_n$, and $(\boldsymbol{\Sigma}_n)_{ij}$ is the ijth entry of $\boldsymbol{\Sigma}_n$. Therefore, we find the covariance matrix M is given by

$$\boldsymbol{\Sigma}_{ij} = \frac{1}{\sum_m w_m} \sum_n w_n \left((\boldsymbol{\Sigma}_n)_{ij} + (\mu_{ni} - \mu_i)(\mu_{nj} - \mu_j)\right). \tag{III.2.22}$$

In matrix form:

$$\boldsymbol{\Sigma} = \frac{1}{\sum_m w_m} \sum_n w_n \left(\boldsymbol{\Sigma}_n + (\mu_n - \mu)(\mu_n - \mu)^T \right). \tag{III.2.23}$$

Then we describe the method to determine which particles should be combined. We choose criteria for combining particles based on their distance being sufficiently close. Since we are more concerned about the computation time for finding combinations rather than have the minimum count of new particles, here we describe a method that scales linearly in the count of particles.

To try combining a particle with others, one can first find all neighbors within a fixed radius, then combine with these neighbors, and iterate through all particles. While apparently finding all particles within a radius threshold needs to compute pairwise distance, it turns out that this fixed-radius near neighbors problem achieves linear scaling in the number of particles [BSW77]. Unfortunately, there is a lack of implementation for these algorithms for dimensions higher than 3. Here we describe a much-simplified version that only finds a subset of all neighbors within a fixed radius since we are only interested in reducing particles quickly, not optimally.

Let the radius threshold for combining particles be r. First, we divide the state space into cubic-shaped buckets with a set radius less than r. Then, we iterate through particles to find which buckets they are in. After that, we iterate through each nonempty bucket and combine particles inside into a single particle. Since the particles are only present in a tiny subset of the buckets, the buckets are initialized in a hash table to take advantage of this sparsity, with their bucket index as keys. While the length of the keys grows linearly with the spatial dimension d, the time taken to evaluate the hash function can be considered as a constant. Under this assumption, the average case writes and access time for a hash table of the occupied bins are practically constant. Hence we achieve practically linear scaling in the number of particles and constant scaling in the number of dimensions if the number of

particles is fixed.

Algorithm 5 Particle Combine

Divide the state space into sufficiently small grids, and find which grid block each particle have its center located (using some Hash function)
for each grid block containing more than one particle **do**
    combine particles with center location defined as (III.2.17), and covariance matrix defined as (III.2.23)
end for

III.2.6: Adaptive timestep particle update and split

When simulating the tissue-level models, it would be ideal if one would not need to worry about the choice of simulation details such as choice of timestep size and focus on the modeling aspect only. This is especially helpful for some biological models, as they can switch between maintaining homeostasis and an abrupt transient behavior, meaning the governing equation is stiff. When solving an ODE, an adaptive timestep solver picks the needed timestep automatically, and we would like to generalize the adaptive solver to the APPD. We notice that in (III.2.10), the equations are ODE. Moreover, the updates of particles are independent, meaning each particle can be updated at a different timestep. Finally, notice that the split of particles is also independent. If we restrict the particle combining and coupling process to some large common timestep τ_0, then in between the common timesteps, all the particles can be updated in parallel, and each particle can be updated with different timesteps that is suitable for it. This section describes how a single particle can be updated between the large timesteps with such an adaptive timestep and splitting.

First, we note that (III.2.10) already formulates the update of a single particle as an ODE, and a standard adaptive ODE solver can be applied if the particle remains small such that the local linear approximation is accurate. Since the method involves a local linear approximation, using a high-order ODE method will not bring any benefit, and we use the Bogacki-Shampine method [BS89] of order 3(2) as the adaptive ODE solver of choice.

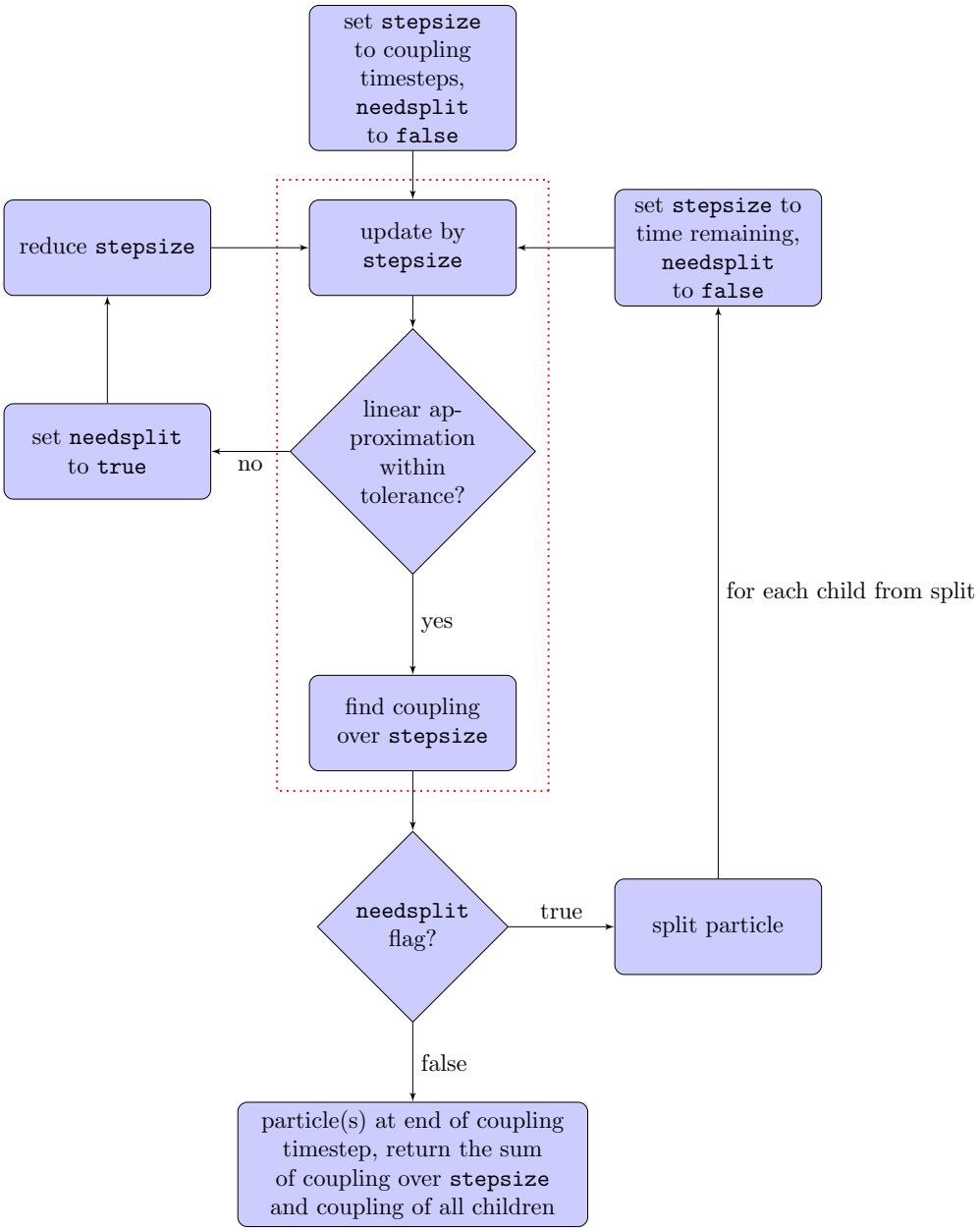


Figure III.3: flow chart for updating a particle, and finding coupling contributed by this particle over a large common timestep τ_0. The red rectangle box suggests a region to factor out if load balancing at particle level is a concern.

It remains to maintain the accuracy of the local linear approximation through large the timestep τ_0. Similar to an adaptive timestep method, we start with a potentially oversized timestep $\tau = \tau_0$. If the timestep τ turns out to be too large, then τ is reduced recursively

until the linear approximation is within the desired tolerance. We update the particle until τ and note that a particle split is needed before the particle can be updated to time τ_0. Consequently, we split the particle in the direction needed, which results in three children particles, as defined in (III.2.15). For each of the children particles, they needed to be updated by $\tau_0 - \tau$. This process is continued recursively until all resulting particles are updated to the common large timestep τ_0. The process of the update is illustrated in the flow chart Fig. III.3.

From the description of the method, we notice that the particles can be updated independently, which gives an embarrassingly parallel problem that can be easily implemented on either a shared memory or a distributed memory system. Since our simulation aims to be executed on a desktop computer or a workstation, we used the `Intel TBB` library to implement this algorithm, which uses a shared memory model.

III.2.7: Extension to nonhomogeneous populations

In a population density method, one constraint for the underlying model is that the elements are assumed to be identical, in that all elements are subject to the same equations. This constraint leaves many interesting scenarios behind, such for a population of neurons with varying intrinsic period. However, we can put the varying property as an additional dimension in the state space, and adjust the formulation correspondingly to accommodate such models for APPD. Specifically, suppose the dynamic velocity of a single element $\mathbf{v}_d$ is defined by both its state variable $\mathbf{x}$ and a constant scalar z:

$$\mathbf{v}_d = \mathbf{v}_d(\mathbf{x}, z) \tag{III.2.24}$$

and an additive noise with diffusion coefficient $\mathbf{K}$. Then, we can include z as an entry in the state space, and define the new state variable

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ z \end{bmatrix}, \tag{III.2.25}$$

with new diffusion coefficient

$$\mathbf{K}' = \begin{bmatrix} \mathbf{K} & \mathbf{O} \\ \mathbf{O} & 0 \end{bmatrix}, \tag{III.2.26}$$

in which $\mathbf{O}$ are zero matrices with appropriate size, and new dynamic velocity

$$\mathbf{v}_d' = \begin{bmatrix} \mathbf{v} \\ 0 \end{bmatrix}. \tag{III.2.27}$$

Then we find $\mathbf{x}'$, $\mathbf{K}'$ and $\mathbf{v}_d'$ satisfies (III.2.1):

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K}' \nabla u - \nabla \cdot (\mathbf{v}' u), \tag{III.2.28}$$

and we can apply the APPD method if a corresponding initial condition is defined. For example, if we assume z follows a normal distribution $Z \sim \mathcal{N}(\mu, \sigma^2)$, and initially $\mathbf{x}$ follows a multivariate normal distribution $\mathbf{X} \sim \mathcal{N}(\mathbf{x}_0, \mathbf{\Sigma}_0)$ that is independent with Z, then the initial condition can be described by a single particle with : **weight** 1, **center location**

$$\mathbf{x}' := \begin{bmatrix} \mathbf{x}_0 \\ \mu \end{bmatrix}, \tag{III.2.29}$$

and **covariance matrix**

$$\boldsymbol{\Sigma}' := \begin{bmatrix} \boldsymbol{\Sigma}_0 & \mathbf{O} \\ \mathbf{O} & \sigma^2 \end{bmatrix}. \tag{III.2.30}$$

III.3: Simulation results

III.3.1: Motivating example: Van der Pol oscillators

First consider a motivating example of a population of Van der Pol oscillators, as an illustration of the method. Recall (III.2.1):

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u), \tag{III.3.1}$$

where, $\mathbf{v} = \mathbf{v}_d + \mathbf{v}_c$. In the case of the Van der Pol model, the oscillator dynamic velocity field $\mathbf{v}_d = \partial \mathbf{x}/\partial t$ is defined by:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \mu(x_1 - \frac{1}{3}x_1^3 - x_2) \\ \frac{1}{\mu}x_1 \end{bmatrix}. \tag{III.3.2}$$

The coupling velocity $\mathbf{v}_c = \mathbf{v}_c(u)$ is defined by

$$\mathbf{v}_c(u) = \alpha \int_{\mathbb{R}^2} y_1 u(\mathbf{y}) d\mathbf{y}, \tag{III.3.3}$$

in which α is a coupling coefficient. This is commonly referred as a *mean-field* coupling. We set up the initial population along the limit cycle of this oscillator without coupling. The results from a sample simulation are shown in Fig. III.4 , which serves as an illustrative example of how the APPD method represents the population and noise over the process of simulation.

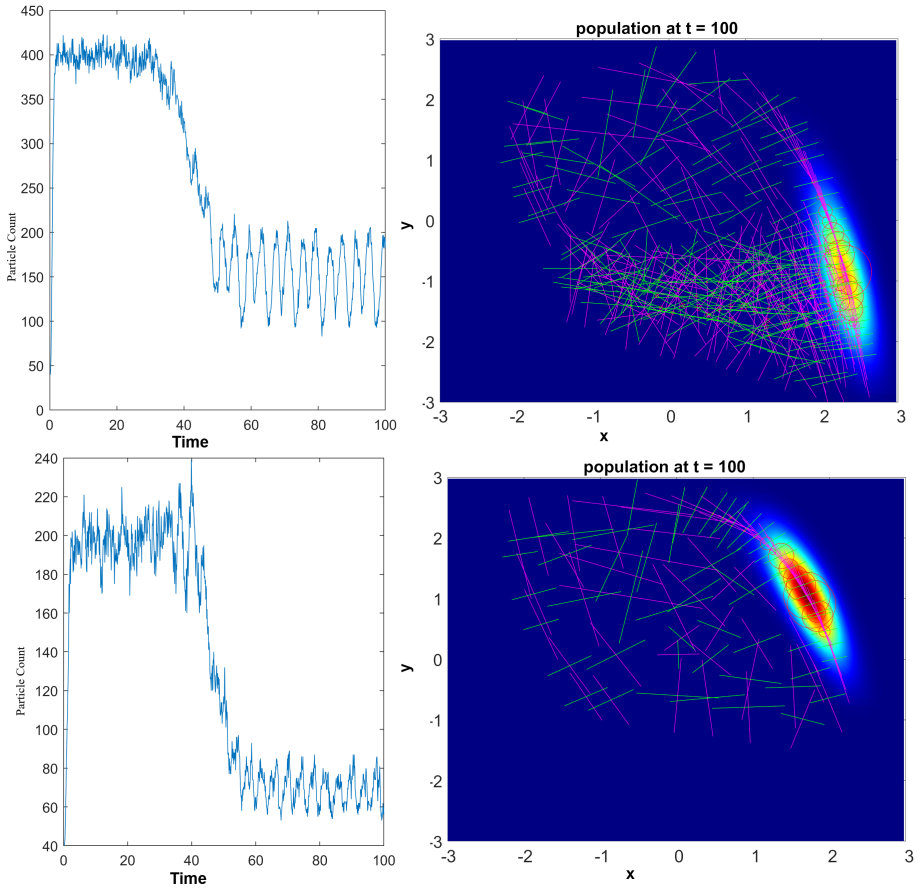


Figure III.4: Motivating example: The population Van der Pol oscillator with $\mu = 1.5, \alpha = 0.5$ and diffusion coefficient $k = 0.1$ (top two) and $k = 0.05$ (bottom two). In the 2 graphs to the right: the red circles indicate the weight of individual particles, the magenta and green lines are the eigenvectors of the covariance matrix with corresponding eigenvalues, while the heat map is the population density at $t = 100$. Note that the color scales of the two right graphs are different. The 2 graphs to the left plot the number of particles versus time. Movies are provided in the supplement.

We choose the Van der Pol oscillator to serve as an illustrative example for the usage of our algorithm, as the state space is 2-dimension and can be plotted without projection. In the example used in Fig. III.4, we select an uncoupled initial condition along the limit cycle and observe the effects of coupling. It can be seen that both populations are coupled;

however, there is a slight phase difference due to the noise level difference. It started at a high value since the population is set to start in an uncoupled state. As the coupling synchronizes the population, the population density is concentrated in a smaller region, and the number of particles drops accordingly. Since the computational cost is proportional to particle count, our method is more efficient if there is stronger coupling and weaker diffusion.

Additionally, we note that synchronization occurs unevenly and that once enough particles start to synchronize, many of the other particles quickly follow suit. This occurs in a similar way to the phase transitions seen in coupled oscillator theory [For17]. As the population synchronizes, the number of particles shrinks.

III.3.2: Kim-Forger circadian model

There is currently much controversy about the role of BMAL1 in the mammalian circadian clock [RVS+20, NCAB21, AGNR21]. BMAL1 is a key activator that binds tightly with repressors to regulate timekeeping. In its absence, BMAL2 has a lower affinity for the repressors, and intracellular rhythms are lost. However, within the network, coupling and noise are thought to interact to still generate rhythms. There is some ongoing controversy about whether these rhythms persist.

To test the role of coupling and noise in mammalian circadian timekeeping, we consider the Kim-Forger circadian model as described in [KF12]. The dynamics of a single oscillator is described by the following equations:

$$\frac{dM}{dt} = \alpha_1 f(P, A, K_d) - \beta_1 M \tag{III.3.4}$$

$$\frac{dP_c}{dt} = \alpha_2 M - \beta_2 P_c \tag{III.3.5}$$

$$\frac{dP}{dt} = \alpha_3 P_c - \beta_3 P \tag{III.3.6}$$

$$f(P, A, K_d) = \left(A - P - K_d + \sqrt{(A - P - K_d)^2 + 4AK_d}\right)/(2A) \tag{III.3.7}$$

We consider a population of these oscillators coupled by their mean value in the repressor

protein P_c, and is also subject to Gaussian noise. Specifically, (III.3.5) is replced by:

$$\frac{dP_c}{dt} = \alpha_2 M - \beta_2 P_c + c_c(\bar{P}_c - P_c), \tag{III.3.8}$$

where c_c is the coupling coefficient, and $\bar{P}_c$ denotes the average P_c in the population. In [KF12], the dissociation constant K_d was investigated in detail for an individual oscillator. It was found that K_d affects both the amplitude and the stability of the oscillation. This is likely a parameter affected by the knockout of BMAL1 as indicated by Fig. 3 in [KF12]. Here, we check how changing this parameterK_d affects the macroscopic behavior of a coupled population.

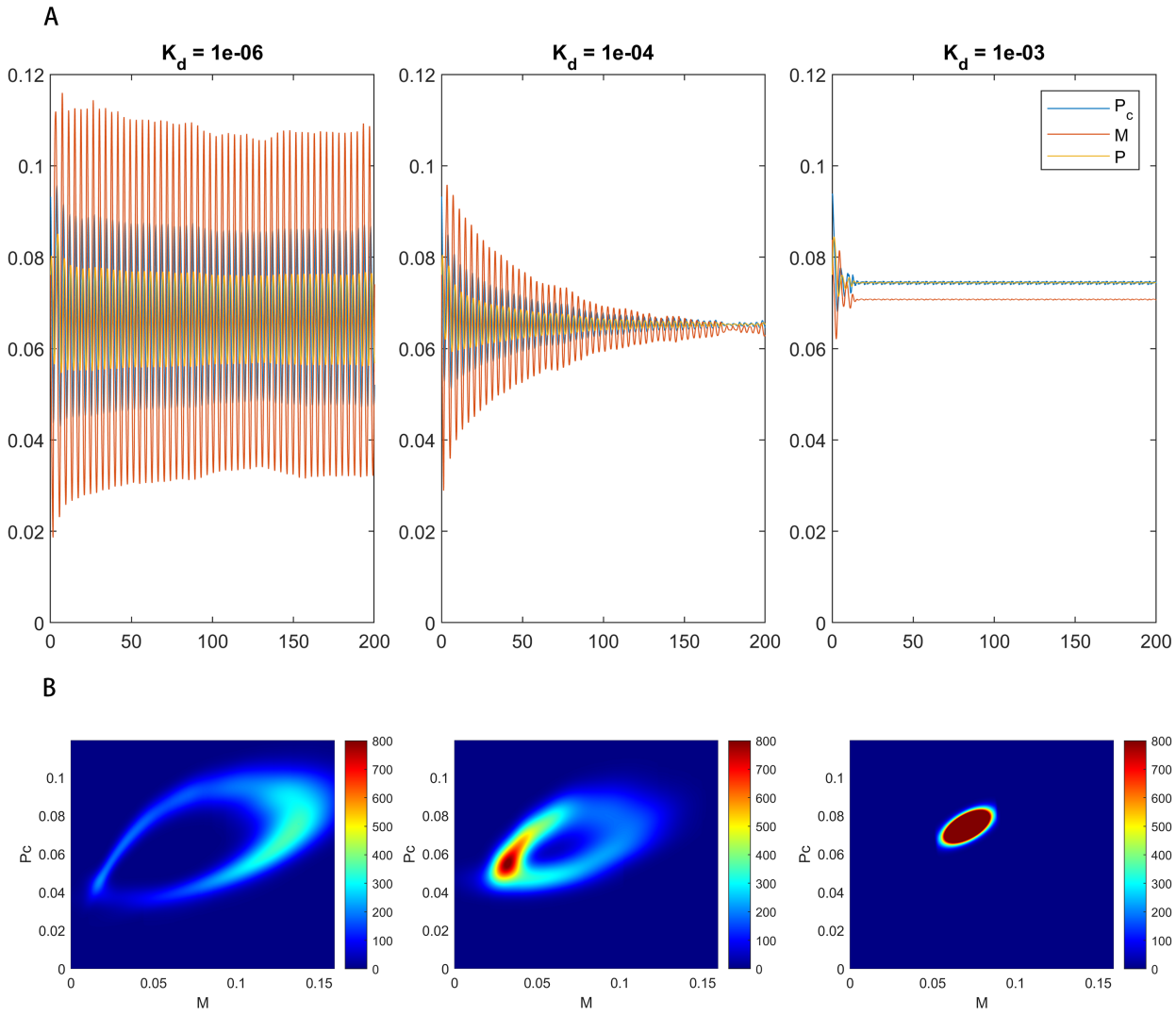


Figure III.5: Predictions of how noise and coupling can generate rhythms in the mammalian circadian clock. **A** shows the average value in P_c, P and M with varying dissociation constant, with $K_d = 1 \times 10^{-6}$, $K_d = 1 \times 10^{-4}$, and $K_d = 1 \times 10^{-3}$ respectively. **B** shows a representative distribution, projected into the $P_c - M$ plane. Other parameters are identical for the 3 instances.

In Fig. III.5, we notice that three patterns emerges from the population as K_d is varied. For a small $K_d = 1 \times 10^{-6}$, [KF12] predicts that the amplitude of the oscillation is large, and

the oscillation is robust. For the coupled population, the average coupling in P_c is sufficient to synchronize the population. For a large $K_d = 1 \times 10^{-3}$, the population stays at a fixed point, which is also consistent with the finding in [KF12] that *sustained oscillations do not occur* under this condition. For an intermediate $K_d = 1 \times 10^{-4}$, the population behaves differently than an individual oscillator: while individual oscillators are in a sustained oscillating state, the coupling is too weak to keep the population synchronized, and the average value for the population reaches a steady state. These simulations teach us two things about the role of BMAL1 in the SCN. First, the nonlinear nature of the coupling is responsible for the BMAL1 KO oscillations in the SCN tissue. This nonlinear neuronal coupling can generate rhythms directly from cells that do not show sustained coupling. Second, the population level rhythms in the SCN are irregular, whereas the rhythms from our simulations are regular. This inconsistency is likely due to the finite-size effect in the SCN, whereas a population density approach assumes an infinite number of elements.

III.3.3: Hodgkin-Huxley model: benchmark

The Hodgkin-Huxley model is a model that describes the electrical activity of a neuron based on ionic currents. Coupled neurons form the basis of computational neuroscience. These models are highly nonlinear and have multiple attractors, which creates challenges for simulation. Additionally, the model exhibits complex behaviors that are highly dependent on the noise level, such as noise-induced synchronization [WCW00, BH06], and a noise-induced coexistence of firing and resting neurons [BNR15]. These interesting macroscopic phenomena require an accurate simulation of noise to reproduce. We use the Hodgkin-Huxley model in two ways. First, we show that the APPD method accurately reproduces the mentioned phenomena, which can be further separated into two cases. Then, we compare it against direct Monte Carlo simulation to show that the method is accurate and fast.

The Hodgkin-Huxley model used here is described as follows: We take $\mathbf{x} = [0.01V, m, n, h]^T$, where V is the membrane potential in millivolts, m, h are proportion(for each cell) of activat-

ing and inactivating subunits of the sodium channel, and n corresponds to that of potassium channel subunits. Here V is scaled by 0.01 such that all the values have the same order of magnitude, where our method is most efficient.

The model equations are as follows:

$$C\frac{\partial}{\partial t}V = -G_{\mathrm{Na}}m^3h(V - V_{\mathrm{Na}}) - G_{\mathrm{K}}n^4(V - V_{\mathrm{K}}) - G_L(V - V_L) - G_{\mathrm{coupling}}(V - V_{\mathrm{coupling}}) - I_{\mathrm{app}} \tag{III.3.9}$$

$$\frac{\partial m}{\partial t} = \alpha_m(V)(1 - m) - \beta_m(V)m \tag{III.3.10}$$

$$\frac{\partial h}{\partial t} = \alpha_h(V)(1 - h) - \beta_h(V)h \tag{III.3.11}$$

$$\frac{\partial n}{\partial t} = \alpha_n(V)(1 - n) - \beta_n(V)n. \tag{III.3.12}$$

With the following equations for the subunit dynamics:

$$\alpha_m(V) = 0.1\frac{V - 25}{1 - \exp(-\frac{V-25}{10})} \tag{III.3.13}$$

$$\beta_m(V) = 4\exp(-\frac{V}{18}) \tag{III.3.14}$$

$$\alpha_h(V) = 0.07\exp(-\frac{V}{20}) \tag{III.3.15}$$

$$\beta_h(V) = \frac{1}{1 + \exp(-\frac{V-30}{10})} \tag{III.3.16}$$

$$\alpha_n(V) = 0.01\frac{V - 10}{1 - \exp(-\frac{V-10}{10})} \tag{III.3.17}$$

$$\beta_n(V) = 0.125\exp(-\frac{V}{80}). \tag{III.3.18}$$

in which G_{coupling} is conductance due to coupling, as explained below.

In a neuron population, it is usually assumed that individual neurons are independent, except when a neuron firing occurs, and post-synaptic neurons are coupled with the firing neuron. In a population density model, the neurons are indistinguishable except for their state. Therefore an all-to-all coupling (or probabilistic coupling) between neurons is assumed.

However, contrary to a neural mass model where coupling strength is determined from the average membrane potential [NT00, DF03], the more realistic threshold coupling can be implemented. Specifically, G_{coupling} is proportional to the flow rate across the hyperplane $V = V_{\text{threshold}}$ from $V < V_{\text{threshold}}$ to $V > V_{\text{threshold}}$ side. Since the threshold value is chosen where the transition is fast, the contribution of diffusion to coupling can be ignored, and the flow rate is given by the velocity of particles multiplied by the marginal density along V. In the actual implementation, we used the averaged flow rate over the previous timestep as the flow rate for the current step, which, considering that actual coupling is not instantaneous, is still a reasonable assumption. The threshold firing potential is chosen at $V_{\text{threshold}} = 45\text{mV}$. It should be noted that the conductance-based coupling is modeled differently than the average-value coupling as introduced in Sec. III.3.2.

Since the subunit variables are proportions of some quantity, their domain is restricted in $[0, 1]$. The dynamics have inward-pointing velocity on the boundary. Therefore the center of particles will not leave the domain without splitting. However, as the particles have volume, the support of centers can be outside this domain due to the effects of Gaussian noise. [AB02] and [KDPN06] discuss more detailed handling of the noise near boundary conditions. Nonetheless, we find that the following alternative method can still produce sufficiently accurate results.

The noise can result in off-domain points in two ways: When a split occurs and when checking an off-center point of a particle in the LSKF method. If a split result in the center of a particle outside the domain, the center is shifted to the closest location inside the domain. If an off-center point is outside the domain while the center is inside, we choose the offset of the point from the particle's center to be its negative, which will be inside the domain and on the level set.

The value of the constants are listed as follows: $C = 1\mu\text{F}\,\text{cm}^{-2}$, $G_{\text{Na}} = 120\mu\text{A}\,\text{mV}^{-1}\,\text{cm}^{-2}$, $E_{\text{Na}} = 115\text{mV}$, $G_{\text{K}} = 36\mu\text{A}\,\text{mV}^{-1}\,\text{cm}^{-2}$, $E_{\text{K}} = -12\text{mV}$, $G_L = 0.3\mu\text{A}\,\text{mV}^{-1}\,\text{cm}^{-2}$, $E_L = 10.613\text{mV}$, and $I_{\text{app}} = 10\text{mA}$. The coupling potential $V_{\text{coupling}} = -35\text{mV}$ for the inhibitory

case, and $V_{\text{coupling}} = 50\text{mV}$ for the excitatory case, keeping in mind that in this original Hodgkin-Huxley model, the neuron rests near 0mV. We choose the diffusion to be homogeneous after V is scaled, with a diffusion matrix $\mathbf{K} = k\mathbf{I}_d$.

The coupling is defined analogously to neuron firing: that is, when a neuron reaches a threshold membrane potential from below, the neuron sends a signal to all post-synaptic neurons. Consequently, G_{coupling} in (III.3.9) is defined by $G_{\text{coupling}} = 20Qc$, where constant c is the coupling coefficient, Q is the flow rate of neurons as a proportion of total population per millisecond. (Therefore, Q has the unit ms^{-1}.)

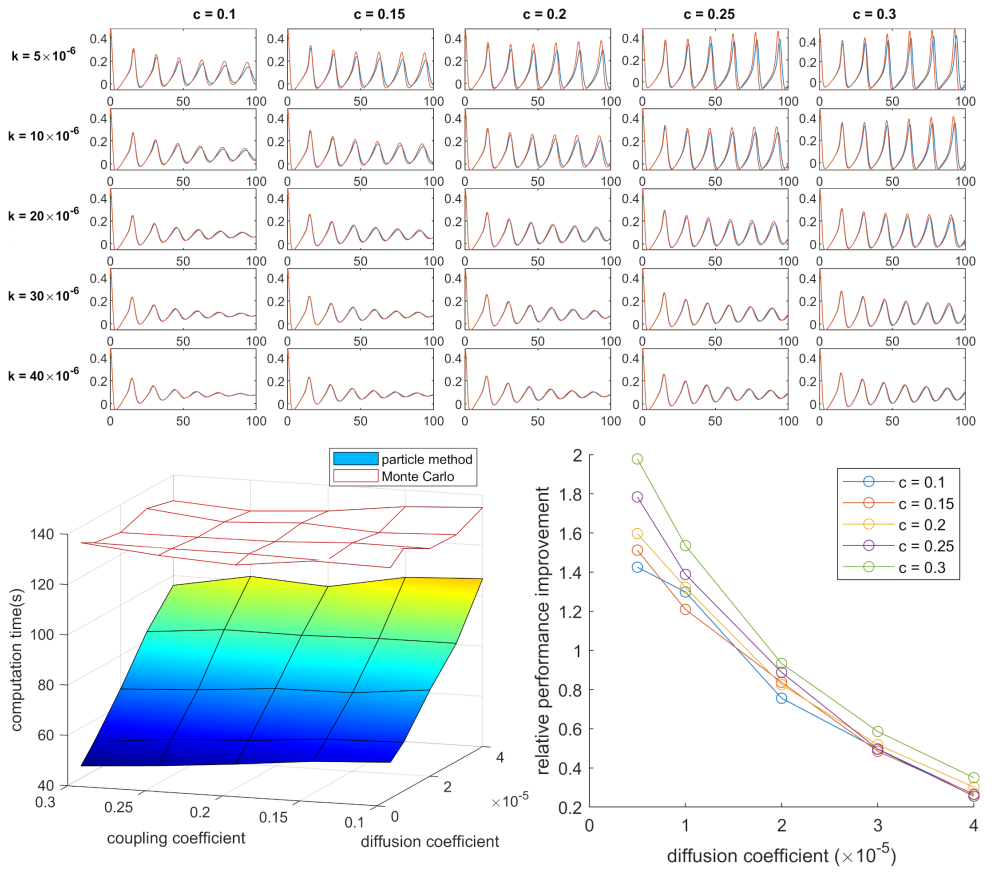


Figure III.6: Testing the accuracy and efficiency of the method. This figure compares the APPD method (blue) result against a Monte Carlo method (orange) with 41080 neurons for excitatory neurons over a range of parameters. The top panel plots the Average membrane potential V(left 5 columns) versus time in microseconds. There are 25 different instances with varying levels of diffusion and coupling strength: From left to right, the coupling coefficient c increases from 0.1 to 0.3 with a stepsize of 0.05. From top to bottom, the diffusion coefficient increases, with $k = 0.5, 1, 2, 3, 4 \times 10^{-5}$ respectively. The lower panel plots the computation time taken for the computation: the left panel shows the computation time, whereas the right panel shows the relative performance improvement over the Monte Carlo method

We compare our simulations to direct Monte Carlo simulations. Note we choose the number of elements $N = 41080$ for the Monte Carlo simulation, and the improvement in

performance would be more significant if a larger number of elements is used. For the range of parameters chosen and our implementation of the Monte Carlo method, the APPD method is faster than the direct Monte Carlo method, as shown in Fig. III.6. It is important to note that while the Monte Carlo method takes about a constant amount of time for all simulation cases, there are large variations for the APPD. At a high noise, low coupling scenario ($k = 4 \times 10^{-5}, c = 0.1$) where the population is asynchronous at the end of the simulation, the APPD method is faster than the Monte Carlo simulation by 20%. Whereas for a low noise strong coupling scenario ($k = 0.5 \times 10^{-5}, c = 0.3$), the population remains synchronized, and the APPD method is 200% faster, as shown in the lower-right panel of Fig. III.6. This again shows that the APPD method is most suitable to simulate neurons with strong coupling at a lower noise level and that the APPD method can adapt to create computational efficiencies.

III.3.4: Hodgkin-Huxley model: macroscopic behavior

The dynamics of the Hodgkin-Huxley model are highly nonlinear and have multiple attractors, which creates challenges for simulation. Additionally, the model exhibits complex behaviors that are highly dependent on the noise level, such as noise-induced synchornization[WCW00, BH06], and a noise-induced coexistence of firing and resting neurons[BNR15]. These interesting macroscopic phenomena require an accurate simulation of both dynamics and noise to reproduce. We show that the APPD method accurately reproduces the mentioned phenomena, which can be further separated into two cases:

1. The applied-current-firing case: Such networks have been predicted to form clusters of neurons that fire together and out of phase with other neurons[TJ00, BH06]. It can be seen in Fig. III.7**A - C** that the neurons split into 3 subpopulations.

2. The noise-induced-firing case: It is predicted[BNR15] and verified with direct simulation[Row07] that the neurons would split into a firing and a resting subpopulation due to the noise

and inhibitory coupling. In the simulation result Fig. III.7**D**, we notice that the average membrane potential increases as the noise level increase, indicating the existence of noise-induced firing of neurons. Whereas in Fig. III.7**E**, we noticed that only a proportion of neurons have fired over a period, which implies that the existence of resting neurons. Additionally, the proportion of firing neurons increases as the noise level is increased.

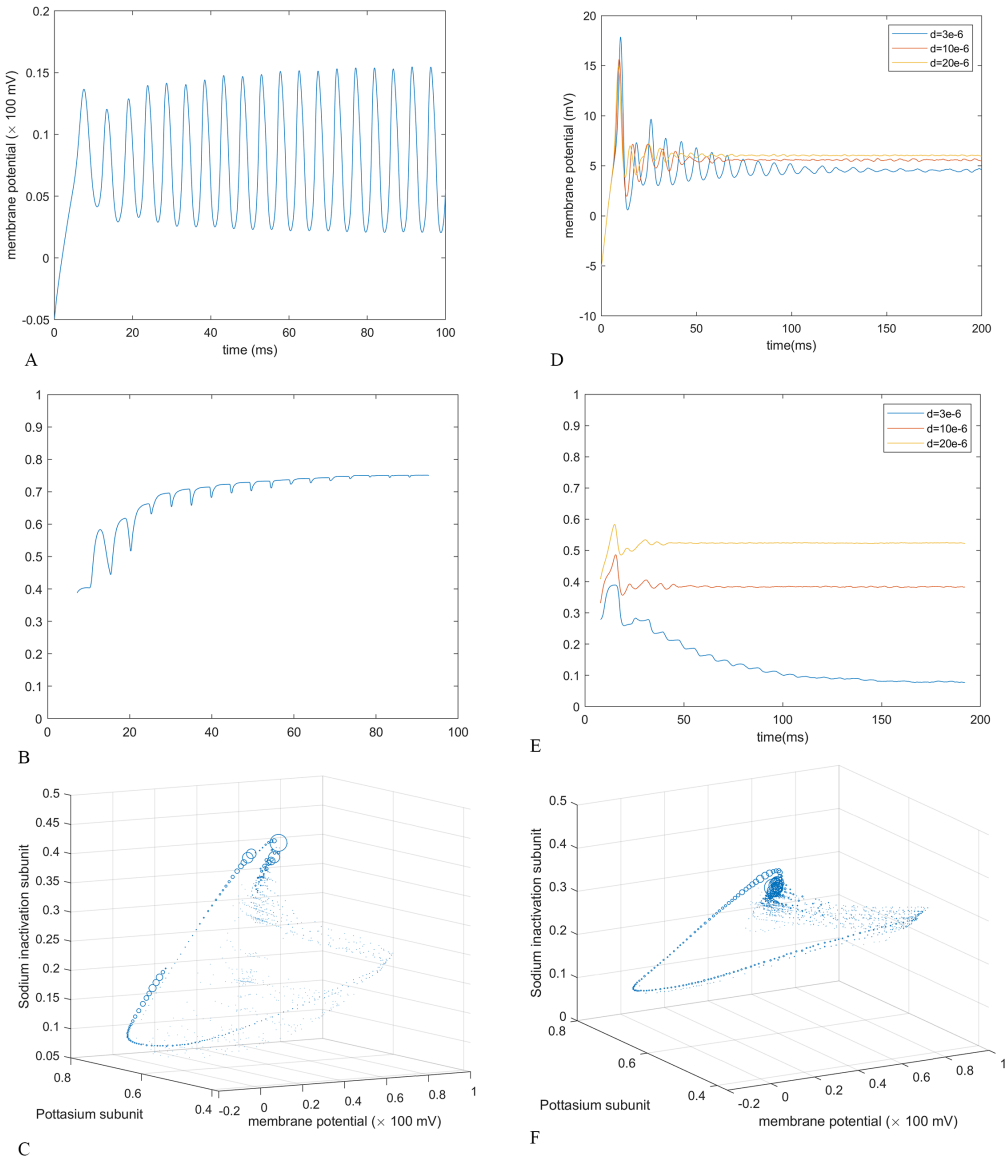


Figure III.7: Population of Hodgkin-Huxley neurons. **A - C** show the applied-current-driven case. **A** shows the averaged membrane potential and the number of particles needed for this simulation versus time. **B** shows that about 70% of the neuron fires per interval. **C** shows the population density projected into the Vh plane at $t = 50ms$ where the neurons split into three subpopulations. **D - F** show the noise-induced firing case under different noise levels, increasing from 3×10^{-6}(blue), to 1×10^{-5}(orange) and 2×10^{-5} . **D** shows the averaged membrane potential, and **E** shows the proportion of firing neurons over the last period for various noise levels. **F** shows that the distribution of particles in the state space.

III.3.5: Hodgkin-Huxley model: non-homogeneous population

Here, we consider a population of identical Hodgkin-Huxley neurons that are otherwise identical except that they are subject to different applied currents, and consequently, different intrinsic periods. This model can be considered as a generalization of the Kuramoto model [Kur03] with much more detailed dynamics and coupling. The equations and parameters are identical to the previous example, except that the applied current is not identical, but given by a normal distribution $\mathcal{N}(6.0\mu\text{A}, \sigma^2)$. Starting with a completely synchronized initial population, we would like to observe how the synchronization varies as the standard deviation and coupling coefficient change.

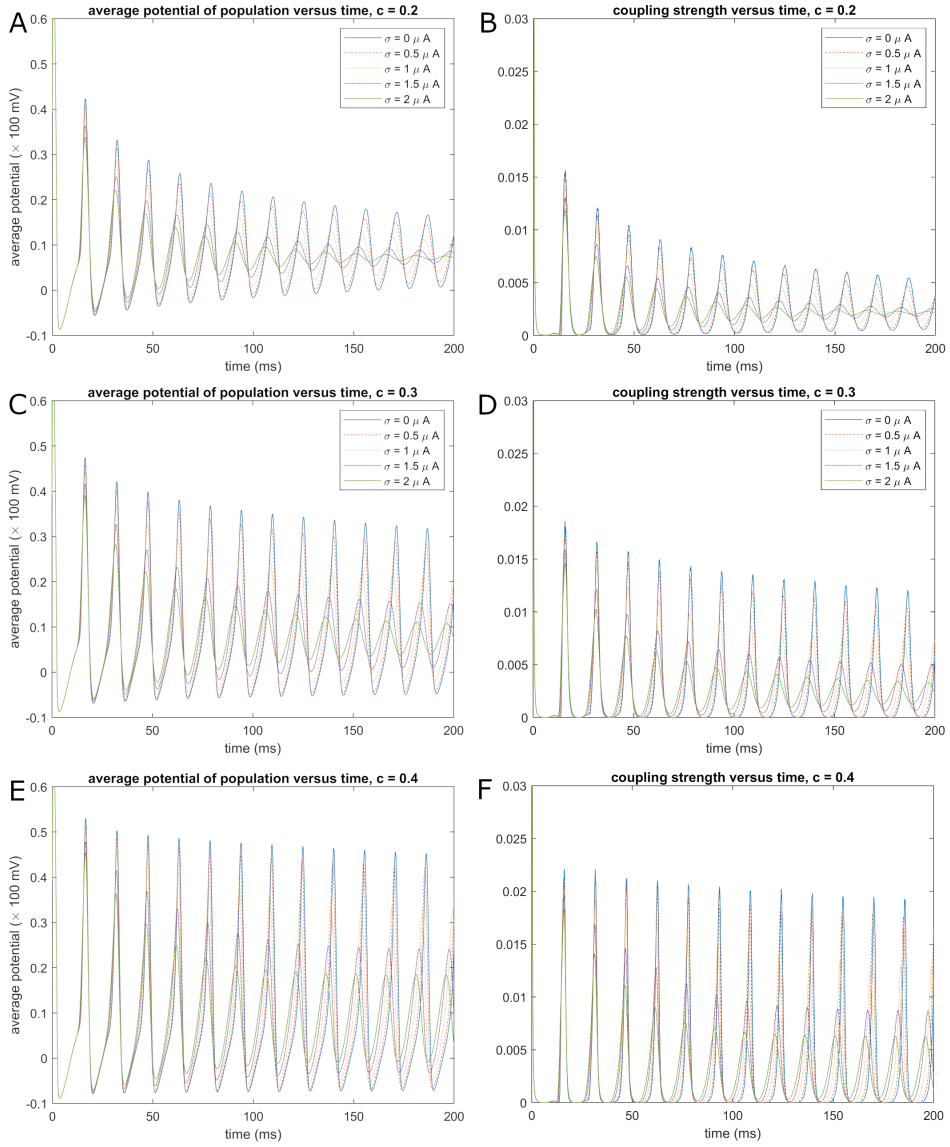


Figure III.8: Retention of synchronization of Hodgkin-Huxley neurons with varying applied currents. Applied current follows the normal distribution $\mathcal{N}(6.0\mu\text{A}, \sigma^2)$. From top to bottom, the coupling increases from 0.2 to 0.4. Inside each panel, the standard deviation of the applied current varies from 0μA to 2μA with a stepsize of 0.5μA. The left panels shows the averaged membrane potential, while the right panels shows the proportion of firing neurons at a given time.

In Fig. III.8, we noted that across all the choice of coupling coefficients, the strength of

the synchronization reduces faster as the variance in applied current increases. As expected, an increased difference in applied current implies a larger variation in intrinsic periods, and the synchronization is harder to maintain. We also notice that the average firing frequency increases as the variance in applied current increases.

III.4: Conclusions

The APPD method provides a fast and accurate method to study population-level behaviors while accounting for noise and coupling in biologically realistic models. We compared our method against direct Monte Carlo simulation and tested its ability to reproduce complex macroscopic phenomena across a range of parameters.

While the APPD method is conceptually inspired by the particle method by Stinchcombe and Forger [SF16], the method presented here has two significant improvements: 1) particles are asymmetric, allowing it to better track the population; 2) computation of the noise term is computed by modifying a single particle instead of interactions between nearby particles, which reduces computation, and allows almost perfect parallel implementation. When compared against existing asymmetric particle methods [Ros96, Ros05, XPP+14, BBH21], our method is the first to be applied to a 4-dimensional problem instead of the typically considered 2-dimensional problems. Such an extension is nontrivial as it depends on the improved numerical stability of our method by tracking a *square-root* instead of the covariance.

Looking at the examples, we checked whether we achieved the goal to develop a middle ground between the direct simulation and existing population density-based methods. From the form of (III.2.10), it is evident that the dynamics of the oscillator are retained, and no manual dimension reduction is required. The choice of the kernel (III.2.5) enabled an accurate representation of the effect of noise. Through the examples, we implemented both all-to-all-coupled oscillators and threshold-coupled oscillators; both resemble the direct simulation.

The advantages of the APPD method are the versatility through different models and a

range of parameters without any model-specific simplifications, and the robustness that the macroscopic behavior matches the direct simulation (Fig. III.6). These observations lead us to conclude that the APPD method can be applied to analyze the complex macroscopic behaviors of a population of coupled noisy oscillators, which is in contrast to many population density approaches that require a model-specific understanding of its behavior to be implemented.

CHAPTER IV

The Mouse Cortex Neuronal Network Model

This chapter is a result of joint work with James Hazelden and Daniel Forger. My contribution includes the generation of connectivities, writing some of the codes used in the simulation, and writing part of the introduction and the methods section of this chapter. James implemented the majority of code in the current version of the software and generated the figures in the result section of this chapter. Daniel supervised our work and helped with the writing of this chapter.

IV.1: Introduction

Perhaps the central question in neuroscience is to understand the mammalian brain at scale. There is a growing consensus that computational methods are an important tool in understanding this multi-scale system. The widely used Hodgkin-Huxley model [HHK52] established the correspondence between the electrical activity and ionic channels of a single neuron [HC97, GB08, YTN16]. This model shows that individual neurons can be capable of surprisingly sophisticated computation. On a higher level, the spatial structure of the mammalian brain is intricate and also important for computation. There is a great need to understand how changes in biophysics at the level of an individual neuron affect whole cortex behavior, especially as genetic, siRNA, or experimental pharmacological techniques can target individual ion channels to determine changes in whole-brain behavior, as measured, for example, by EEG or fMRI [MMK+21, MSS+19].

However, simulating large regions of the brain is computationally intensive. Many existing methods either use a mean-field method that does not resolve individual neurons [SLKW+13, MWJB17], or use a simplified single-neuron model that does not resolve the ionic channels [Mar06]. These simplifications limit the predictions made with such a model, particularly how changes in ionic conductances of individual neurons affect overall behavior. A detailed survey of neurons inside the mouse brain has become a requirement for a convincing cortical model. While previous simulations of whole mouse-cortex [MWJB17] used a probabilistic model, the CUBIC-Atlas [MMS+18] is the first to map the coordinate of all neurons inside a mouse brain and provides the foundation to develop the first whole mouse-cortex model that resolves single neurons. Integrating the CUBIC-Atlas with an existing high-resolution data-driven connectivity model [KHG+19] and the distribution of inhibitory and excitatory neurons [EGKM18], we established a neuronal connectome model with up to all 13×10^6 neurons [MMS+18] inside the mouse cortex based on experimental results.

To efficiently simulate this large-scale network with up to 13×10^6 neurons, we developed our simulation framework in `CUDA` to take advantage of the efficiency of GPU in computing highly parallel workloads. We have carefully optimized our implementation such that we can achieve comparable performance or exceed the performance of state-of-the-art software such as Brian [GB08], NEURON [HC97] and GeNN [YTN16] on a single desktop GPU.

As important as an efficient simulation method is a versatile data visualization tool. Data visualization is designed as an integral part of our software framework. We implemented a real-time visualization of individual neurons using `OpenGL`, allowing the observation of propagation of neuronal waves on the CUBIC-Atlas-mapped neurons while the simulation is running. Running on a desktop computer is a *feature* rather than a *compromise* as such a real-time visualization would not be possible via `SSH`, which is popular with computer clusters. Alongside the real-time visualization, we can also give realistic simulated event-related potential (ERP), and electroencephalography (ERG) traces measured from used-defined regions with the detailed spatial information provided in the CUBIC-Atlas.

With the aid of the software framework and the data-driven model, we demonstrate that how factors in the model such as connectivity, neuron count, individual neuron ionic dynamics, and mixing different neuron types can change properties of the system, such as cortical rhythms, formation of oscillations, and traveling planar or spiral waves. Taken together, we find that the interplay between the detailed structure of the mammalian brain and sophistical information processing by a single neuron generates behavior that cannot be predicted at a single scale.

IV.2: Methodology

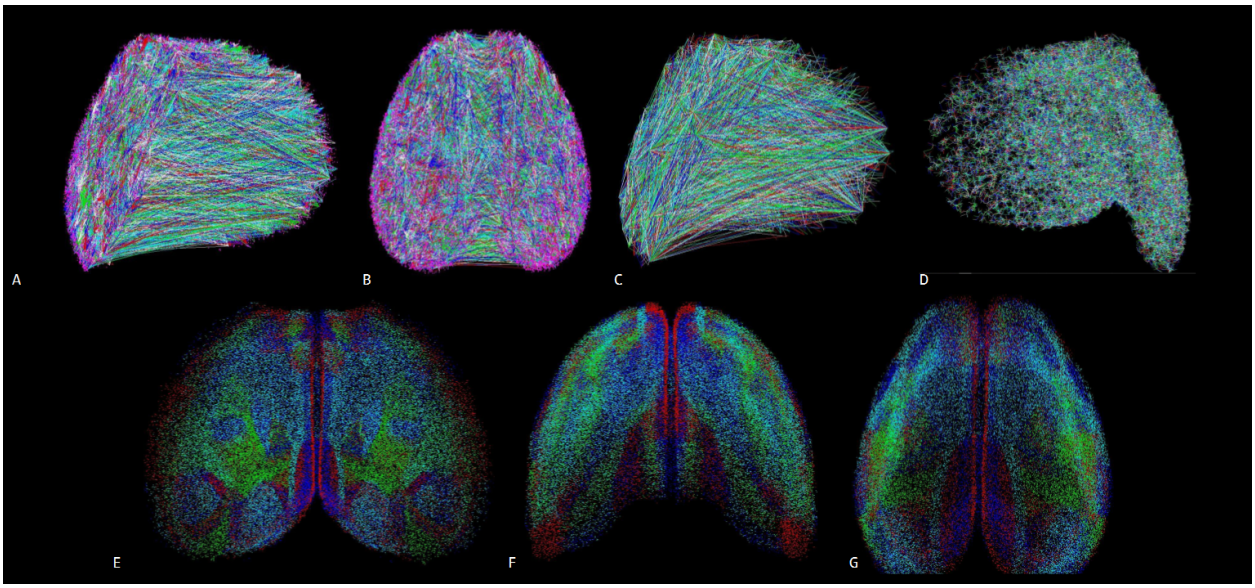


Figure IV.1: **This figure shows information related to the connectivity. A - D** illustrate the connectivity used in our mouse brain model by plotting all downstream connections for 1000 randomly chosen neurons. Panel **A** and **B** show the AMPA long-range "white matter" connections. Panel **C** shows the inhibitory GABA connectivity. As can be seen, it is sparser than the AMPA connectivity. Panel **D** shows an example of a generated short-range "grey matter" connectivity. Panel **E** through **G** color 10^6 neurons based on the region to which they belong. The neurons are visualized from the front, back, and top of the mouse brain, respectively. We used intracranial electrodes from a clinical study and projected these onto the brain by voxelizing the point cloud to make the projection.

IV.2.1: The model for a single neuron

The Hodgkin-Huxley model [HHK52] is a model that describes the activity of a neuron based on the dynamics of individual ionic currents. In the implementation that follows, we use the Hodgkin-Huxley model with parameters fitted for cortical models, as presented

in [For17]. We take $\mathbf{x} = (V, m, n, h)$, where V is the membrane potential in millivolts, m, h are proportion(for each cell) of activating and inactivating subunits of the sodium channel, and n corresponds to that of potassium channel subunits.

The model is described by the following ordinary differential equations (ODE):

$$C\frac{\partial}{\partial t}V = -G_{\mathrm{Na}}m^3h(V - V_{\mathrm{Na}}) - G_{\mathrm{K}}n^4(V - V_{\mathrm{K}}) \tag{IV.2.1}$$
$$- G_L(V - V_L) - G_{\mathrm{AMPA}}(V - V_{\mathrm{AMPA}}) - G_{\mathrm{GABA}}(V - V_{\mathrm{GABA}}) - I_{\mathrm{app}}$$

$$\frac{\partial m}{\partial t} = \alpha_m(V)(1 - m) - \beta_m(V)m \tag{IV.2.2}$$

$$\frac{\partial h}{\partial t} = \alpha_h(V)(1 - h) - \beta_h(V)h \tag{IV.2.3}$$

$$\frac{\partial n}{\partial t} = \alpha_n(V)(1 - n) - \beta_n(V)n. \tag{IV.2.4}$$

in which G_{AMPA} and G_{GABA} are conductance due to coupling input from other neurons. With the following equations for the subunit dynamics:

$$\alpha_m(V) = 0.1\frac{V - 25}{1 - \exp(-\frac{V-25}{10})} \tag{IV.2.5}$$

$$\beta_m(V) = 4\exp(-\frac{V}{18}) \tag{IV.2.6}$$

$$\alpha_h(V) = 0.07\exp(-\frac{V}{20}) \tag{IV.2.7}$$

$$\beta_h(V) = \frac{1}{1 + \exp(-\frac{V-30}{10})} \tag{IV.2.8}$$

$$\alpha_n(V) = 0.01\frac{V - 10}{1 - \exp(-\frac{V-10}{10})} \tag{IV.2.9}$$

$$\beta_n(V) = 0.125\exp(-\frac{V}{80}). \tag{IV.2.10}$$

The value of the constants are listed as follows: $C = 1\mu\mathrm{F\,cm^{-2}}, G_{\mathrm{Na}} = 120\mu\mathrm{A\,mV^{-1}\,cm^{-2}}, E_{\mathrm{Na}} = 115\mathrm{mV}, G_{\mathrm{K}} = 36\mu\mathrm{A\,mV^{-1}\,cm^{-2}}, E_{\mathrm{K}} = -12\mathrm{mV}, G_L = 0.3\mu\mathrm{A\,mV^{-1}\,cm^{-2}}, E_L = 10.613\mathrm{mV}, I_{\mathrm{app}} = 10\mathrm{mA}$. The coupling potential $V_{\mathrm{GABA}} = -70\mathrm{mV}$ for the inhibitory case, and $V_{\mathrm{AMPA}} = 0\mathrm{mV}$

IV.2.2: Synaptic coupling between neurons

In our model, the synaptic connections between neurons are assumed to be fixed. To represent the connections between neurons for a neurotransmitter (in our model, AMPA and GABA are considered), we index the neurons by an integer. Then, we can represent the connectivities between neurons as a connectivity matrix $\mathbf{M}_c$, where $(\mathbf{M}_c)_{ij}$ is 1 if there is a connection from the presynaptic neuron i to the postsynaptic neuron j, and is 0 if otherwise. Since each neuron has about 10 to 1000 postsynaptic neurons depending on the model assumption (detailed in Sec. IV.2.4), the matrix is sparse. To address the sparsity, it is expressed in the compressed row format. We assume that a presynaptic neuron fires after its membrane potential V increases above a threshold potential and that the time taken for the postsynaptic neuron to receive this firing signal can be neglected.

IV.2.3: Updating neurons in time

To update the states of the neurons in a network in time, both the dynamics of the neuron and the synaptic coupling need to be computed. To separate these two operations, we update the neurons by the dynamics for a small timestep and assume that all synaptic couplings occur at the end of the timestep.

For the update between the timesteps, note that the Hodgkin-Huxley model is described by ODEs; therefore, they can be updated by a variety of numerical integration methods. In our implementation, we used the leapfrog method. Since the neurons are independent, this update is embarrassingly parallel.

For the synaptic coupling at the end of the timestep, we can record the firing neurons using a vector $\mathbf{v}_c$: $(\mathbf{v}_c)_i$ is 1 if the neuron i fired during this timestep, and 0 otherwise. v_c is sparse most of the time, as the timestep taken is small. Then, the effect on the postsynaptic neurons can be founded by the matrix-vector multiplication $\mathbf{v}_c^T\mathbf{M}_c$.

To ultilize the sparsity in both $\mathbf{M}_c$ and $\mathbf{v}_c$, the matrix-vector multiplication is computed as follows:

- Collect the index of nonzero terms in $\mathbf{v}_c$, store the indices as a vector $\mathbf{I}_c$
- Then $\mathbf{v}_c^T \mathbf{M}_c$ is found by adding rows with indices $\mathbf{I}_c$ in $\mathbf{M}_c$.
- Since $\mathbf{M}_c$ is represented in the compressed row format, collecting all the corresponding rows in $\mathbf{M}_c$, we get a sequence of indices (that can be repetitive) where each index corresponds to a firing at the postsynaptic neuron indexed.
- Starting with a 0 vector of length the number of neurons, adding 1 to each of the indices results in the effect on the postsynaptic neuron. This process can be done in parallel, as long as a concurrent data structure for the vector is used to avoid racing conditions caused by the repetition of indices.

IV.2.4: Generating a realistic connectome from the Allen Brain Atlas and the CUBIC-Atlas

We consider two types of synaptic connections: the *local connectivity* that models the majority of connections (90%), and the *global connectivity* that generates long-range connectivity between different regions of the brain. The local connectivity is generated statistically by assuming the probability of connectivity between pairs of neurons decays exponentially as a function of the distance between the neurons. Additionally, we only consider local pairs of neurons that are closer than a *threshold distance* since they exclude a negligible number of connections but greatly saves on computation time. Global connectivity is based on a high-resolution model of the mouse connectome [KHG+19], which is a further development from [OHN+14] and is based on stereotaxic injection.

In generation of *global connectivity*, we used the *voxel-based* connectivity model proposed in [KHG+19]. This connectivity model divides the mouse brain into small cubes, called *voxels*, that form a three-dimensional grid. [KHG+19] experimentally finds the density of connectivity between voxels. To generate the *global connectivity*, we start with the coordinates of neurons in CUBIC-Atlas [MMS+18], perform a coordinate transformation to CCF

v3 [all], then find the voxel that contains this neuron. The CCF v3 is a standard coordinate system for mouse brains that accommodates the difference in shape between individual mouse brains. To determine postsynaptic neurons, we treat the density of connectivity from source voxel to target voxel as the probability of connections and that the neuron has an equal probability of connecting every neuron within the same target voxel. A detailed description of global connectivity generation is included in the appendix. Both local and global connections are shown in Fig. IV.1

IV.2.5: Identifying excitatory and inhibitory neurons

We consider excitatory and inhibitory neurons, depending on the type of neurotransmitters released. The distribution of excitatory and inhibitory neurons is heterogeneous in space, and the expression level of certain genes such as NRN1 (excitatory) and GAD67 (inhibitory) can be used as a proxy to approximate this distribution [EGKM18]. In their work, a distribution of inhibitory neurons in different regions of the brain is derived from a gene-expression atlas [LHA+07] under the assumption that 7.94% of the neurons are inhibitory [KYP+17]. From the CUBIC-atlas [MMS+18], we determine the brain region each neuron belongs to and randomly decide whether the neuron is inhibitory based on this conditional probability for its brain region.

IV.3: Results

IV.3.1: Emergence of Travelling Cortical Oscillations

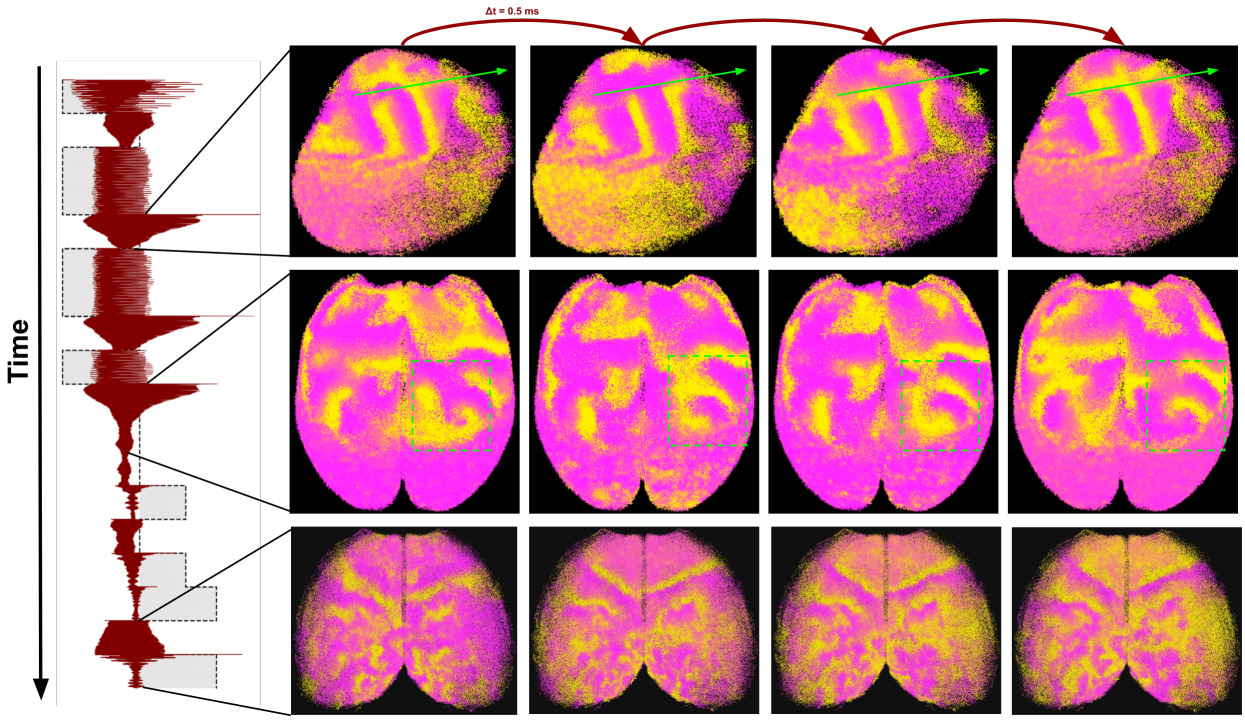


Figure IV.2: Spatial behavior of the whole cortex model with varied periodic input stimuli. The left plot shows the normalized firing rate of the whole cortex model over a continuous simulation timeframe. The blue trace shows the incremented input that was applied in this case to the visual regions of the mouse cortex model. In particular, all regions of the mouse cortex were provided with base stimuli, and the visual region was given extra input stimulation based on this blue plot. As can be seen, the input is periodically applied every second and alternates on-off with a frequency of 50Hz. On the right of the figure, screenshots of our visualization of each individual neuron's voltage are shown, demonstrating the evolution of the model during each timeframe highlighted.

The mammalian brain exhibits a number of important oscillations of different frequencies that can emerge in a variety of situations. For example, during sleep, delta waves travel across the mouse cortex; the reason for such waves and their impact on a variety of phenomena such as memory, synaptic plasticity, and regulation of concentrations of different chemicals in the brain is a topic of much research. To exhibit spatial dynamics in the brain, individual dynamics of many neurons or small populations of neurons must be simulated. A number of papers have done such simulation, usually using some combination of short-range connections and uniform random connections on a collection of grids. One downside to this approach, however, is that morphology of the brain is not realistically accounted for. It is clear that waves will travel differently through a population of neurons placed on a uniform grid versus on a realistic model of the mouse brain morphology. Factors such as wave speed and size are determined by the morphology. Our model of the mouse brain thus simulates individual neurons with realistic brain morphology.

Fig. IV.2 demonstrates the emergence of three types of whole-brain phenomena as a consequence of varied input stimuli applied to the visual regions of the mouse cortex. In this case, the Hodgkin-Huxley model was used to model neurons in the sleeping state. Stimulation was modeled simply as an input current added to the voltage derivative. In the simulation shown, the localized connections in the mouse grey matter were given a high conductance versus the long-range connections, and thus large-scale waves can be observed. It should be noted, however, that when the long-range connection conductance is made, larger spatial patterns are more local and more disrupted (see Section IV.3.2 below for more analysis of this relationship).

Chaotic desynchronized firing with no clear noticeable patterns, as shown in the top right of Fig. IV.2 is typical in our simulations and for real brain dynamics. Such firing will persist in a steady cycle behavior unless input current or the dynamics of the brain are modified during simulation. In this desynchronized case, approximate fMRI of the whole brain takes the form of an almost constant line, as can be seen on the top left of Fig. IV.2.

A more complex situation emerges when the brain is provided with non-uniform input stimuli. Such a situation is shown in the middle right part of Fig. IV.2, demonstrating traveling waves in our cortical model. These waves originate at the point of stimulation–in this case, the visual regions which reside in the anterior, dorsal regions of the mouse cortex. The frequency of these waves is high ($f > 30$Hz gamma waking oscillations), as is expected for neurons during the wake.

Even more non-trivial spatial dynamics can be observed in our model when neurons are provided with a high level of stimulation. In particular, when we increase the input current to our model to high levels, spiral radiating waves emerge in the regions of high stimulation, as shown in the bottom right of Fig. IV.2. The spiral wave pattern is an area of experimental investigation, and it has been found that input current can be used as a "switching" mechanism between desynchronized firing, planar waves, and spiral waves, as is consistent with our results [Jal03, HXL+10]. A number of authors have attempted to model spiral waves in the cortex by PDEs, such as [BLS95]. Because of the difficulty involved in attaining experimental data exhibiting such spiral waves, computational models such as ours that allow one to easily exhibit such waves and investigate the role of individual ion channels and other factors in their emergence are suitable for aiding in the analysis of spiral waves in the cortex.

IV.3.2: Impact of Gray and White Matter Connectivity and Number of Neurons on Oscillations and Synchrony

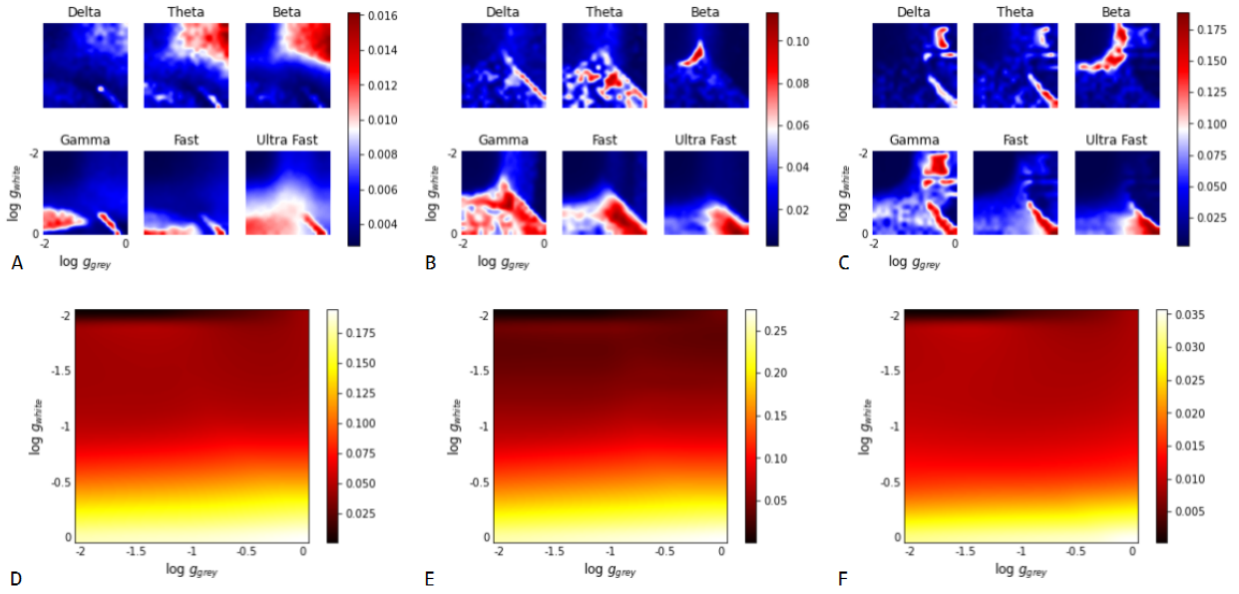


Figure IV.3: Gray and white matter parameter sweeps with varied numbers of neurons. Pairs **AD**, **BE**, and **CF** show results for simulations using 10^4, 10^5, and 10^6 neurons, respectively. White and gray matter synaptic strengths were uniformly varied logarithmically from 10^{-2} to 10^0 in each case. **A - C** show the frequency power spectra of simulated EEG using our mouse brain model simulated for an 8-second time frame with an initial transient simulation of 4 seconds. 1000 neurons were sampled from the whole cortex randomly to generate the ERP used to measure power spectral density. **D - F** plot the corresponding synchrony for the parameter each parameter sweep shown in **A - C**. 4096 neurons were sampled to measure synchrony.

Two large-scale measures of brain activity are power spectral density, computed using the fast Fourier transform of an ERP signal in a certain part of the brain, and whole-brain synchrony, as described in Section IV.3.4. We used these metrics to evaluate how changing global synaptic strength for long-range connections passing through the brain white matter

versus short-range connections passing through the brain gray matter affects whole-brain activity. Furthermore, we performed these experiments with a varied number of neurons with a fixed spatial extent of the localized short-range gray matter connections and by selecting a subset of the total white matter connectome. We performed a parameter sweep varying white and gray matter synaptic strengths logarithmically from 10^{-2} to 10^{0} and computed the whole brain synchrony using 4096 randomly chosen neurons from the cortex and broke down the power spectral density into six groups.

Fig. IV.3**A - C** show the frequency power spectra of simulated EEG using our mouse brain model. As can be seen, the power spectra vary significantly based on the number of neurons used and do not exhibit any simple patterns. In each case, most of the power is in the gamma through ultra-fast frequency ranges, as is expected because the Hodgkin-Huxley model was used. From these plots, it is clear that both long and short-range connections have a non-trivial effect on oscillation formation in the cortex and that as a number of neurons are varied because realistic connectivity was used, this relationship changes. Furthermore, it has been found experimentally that lower frequency oscillations should be more significant than higher frequency oscillations, but we do not see this in many cases in the above plots. Thus, we can also conclude that there are other mechanisms in the brain other than synaptic structure and strength that generate such low-frequency oscillations.

Fig. IV.3**D - F** plot the synchrony of 4096 randomly chosen neurons according to the parameter sweep and with varied N for each individual panel. In contrast to the power spectra, although the maximum synchrony changes based on the number of neurons, the dependence on the gray and white matter is almost the exact same. From the latter three plots, we conclude that synchrony is not significantly impacted by the number of neurons and that as the strength of long-range connections increases, the synchrony of the whole waking brain also increases. Furthermore, short-range connections have little impact on the whole-brain synchrony. This agrees with the synchrony plots in Fig. IV.5: removing localized connections had little impact on synchrony, but replacing the long-range connections drastically affected

synchrony.

IV.3.3: Emergence of Functional Connectivity from Physical Connectivity

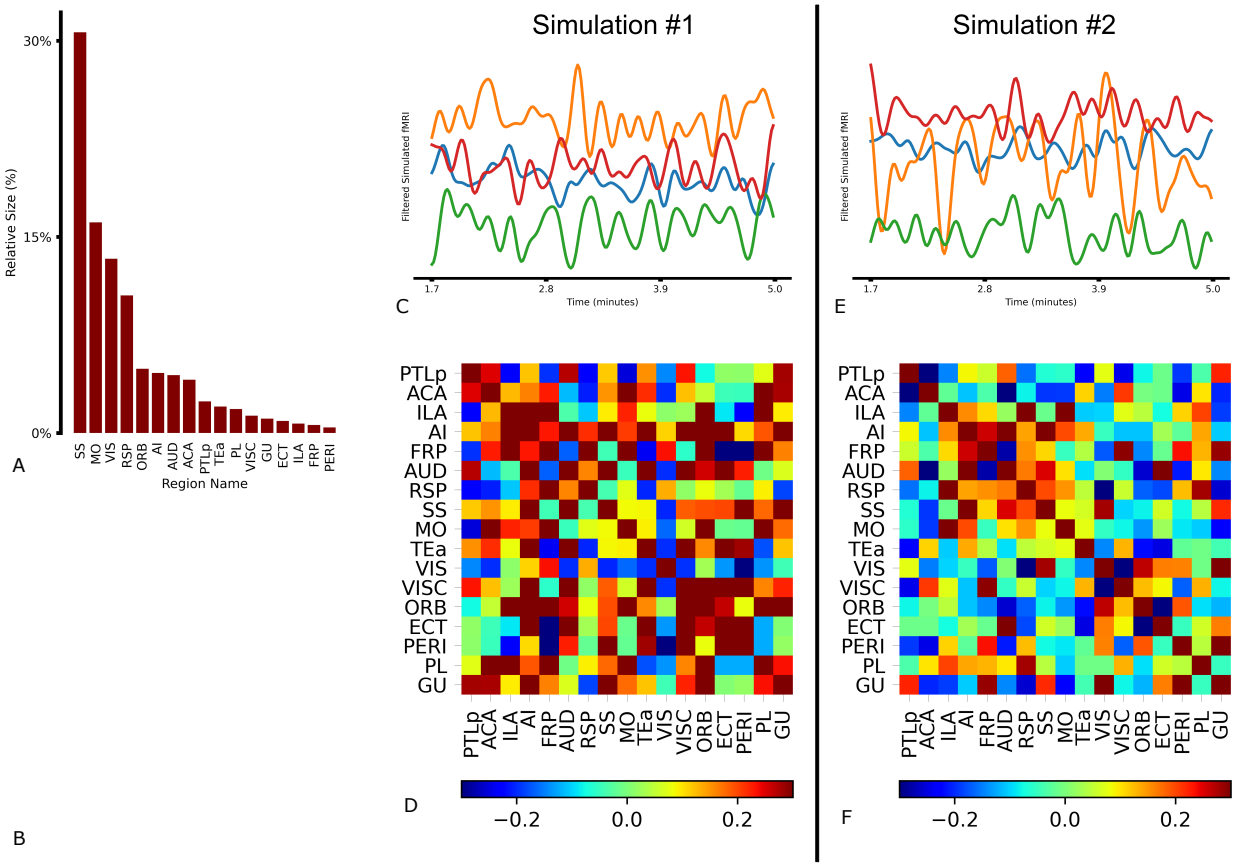


Figure IV.4: Evaluation of the functional connectivity of the network. **A** shows the percentage of neurons in each region of the cortex. **B** shows the number of synapses between regions. **C - D** and **E - F** shows results from two distinct simulations. In both simulations, the visual regions were stimulated more than all other regions, which were stimulated uniformly. **C** and **E** show result from simulated fMRI, whereas **D** and **F** show the functional connectivity between regions.

One avenue of use for our simulation framework is the evaluation of functional connectivity of the mouse cerebral cortex. Such connectivity is measured as the temporal correlation

between activity in different regions of the mouse cortex. Functional connectivity is different from physical connectivity, which is simply the physical synaptic connections in the network of neurons. This physical connectivity between regions is plotted in Fig. IV.4**A - B**. **C - D**, **E - F** shows results of functional connectivity with 2 random distinct initial conditions for neurons.

Most commonly, to measure experimental functional connectivity, fMRI is used to measure activity. To measure the correlation between two fMRI traces, it is common to first filter the fMRI signals to a low-frequency band, and then the correlation between two regions is computed as the Pearson correlation between the two corresponding filtered fRMI traces. In the simulated fMRI, the average membrane potential is first measured over a 5-minute timeframe, then bandpass filtered in the range 0.008 Hz to 0.8 Hz. We reproduced this approach with the Allen brain primary subregions of the isocortex. First, activity approximating fMRI was measured by measuring the firing rate for each region over a 5 minute time interval. 10^6 neurons overall were used. The firing rate traces were then low-pass filtered to the 0.8 Hz frequency band using a Butterworth filter. The resulting fMRI approximation and the low-pass filtered fMRI are shown in Fig. IV.4**C** and **E**. As can be seen, although the fMRI and functional connections are similar, there are clear differences. This suggests that functional connectivity will naturally vary somewhat between experiments on the same specimen. One issue yet to quantify is how much functional connectivity can vary with the same setup but random initial conditions. This is an important question because we would like to distinguish between variations resultant from large-scale differences in functional connectivity and variations resulting simply from different initial conditions.

IV.3.4: Realistic Connectivity Significantly Impacts Simulated ERP

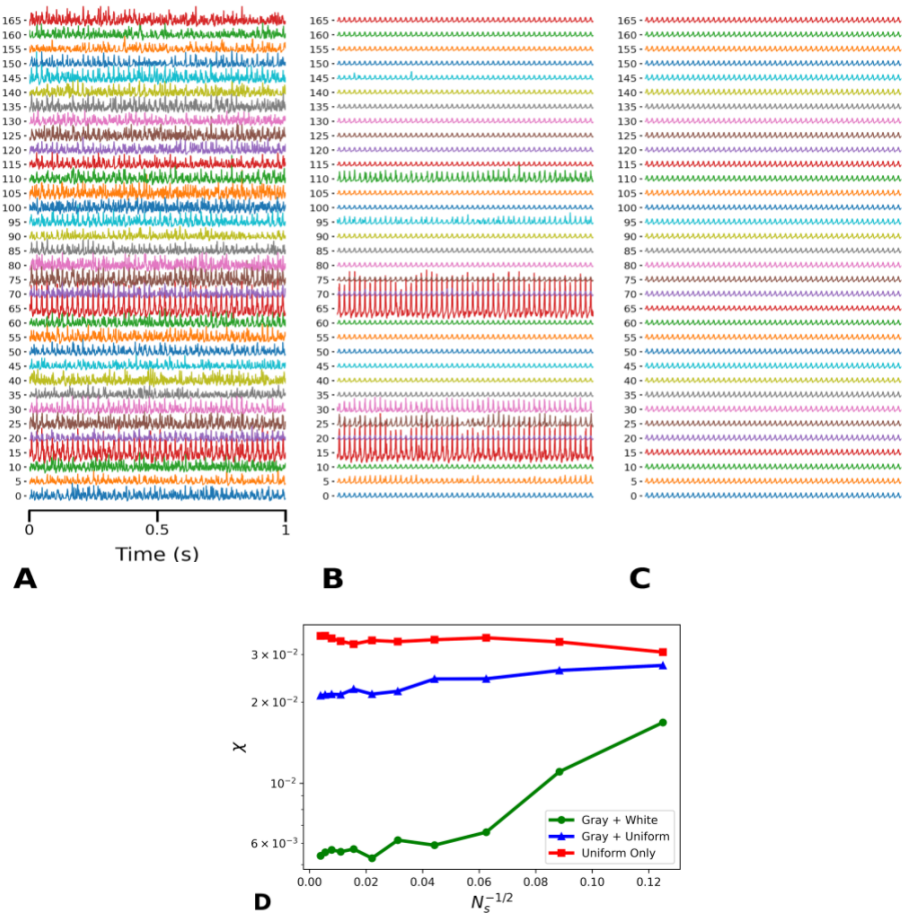


Figure IV.5: Three different connectivities were used to demonstrate the importance of realistic connectivity for EEG and synchrony. **A - C** show simulated ERP (event-related potential) for every five CUBIC regions. **D** shows the plots of the synchrony of the whole network by sampling N_s neurons uniformly from the cortex and computing the variance of the network average divided by the average individual variances. Multiple choices of N_s are shown to demonstrate the trend as the sample size is increased.

To evaluate the importance of realistic physical connectivity based on a real physical morphology, we compared three different scenarios with different excitatory AMPA connectivities (with the same fixed long-range GABA connectivity generated from the Allen brain atlas).

In particular, in the first scenario, the realistic whole-cortex connectivity generated from the Allen brain atlas with short-range "gray matter" connections was used. In the second scenario, the long-range AMPA connections were replaced by uniformly randomly distributed connections between all the neurons. Finally, in the last scenario, the short-range connections were removed, and only the uniform random connections were used to simulate AMPA dynamics.

Fig. IV.5**A - C** show the ERP signals of simulations run in each of these three scenarios, respectively. In particular, for each region in the CUBIC atlas with at least 100 neurons, the average voltage of 100 uniformly chosen neurons was measured. Every five such traces is plotted for each connectivity scenario. Simulations were performed with fixed initial conditions for all neurons and no noise. Furthermore, all neurons in the particular CUBIC region with the highest number of neurons were stimulated with a fixed input current to induce firing. Finally, the cortex was simulated for a full transient second before making measurements.

As can be seen, Fig. IV.5**A** demonstrates complex dynamics resembling observable experimental ERP traces during the wake for the mouse brain. Fig. IV.5**B**, on the other hand, lacks much of these complex traces. However, some of the traces from **A** are almost the same in **B** such as trace 65 and 15, suggesting that these traces were mostly influenced by short-range gray matter connections. Finally, C demonstrates that in the case of just uniform random connectivity, there are almost the exact same firing patterns for the ERP average voltage for each region. Thus, these simulated ERP traces suggest that uniform random connections cannot be used to accurately capture the nuanced behavior of the real mouse brain across regions, even when short-range connections are introduced.

This conclusion is further supported by Fig. IV.5**D**, which plots synchrony of the whole mouse cortex for each of the three connectivity scenarios. Fig. IV.5**A** shows the EEG trace with both realistic gray and white matter connections. In case Fig. IV.5**B**, the realistic long range white matter connections were replaced with uniform random connections. As can

be seen, some patterns from **A** are preserved, but most of the signals are more uniform. Finally, in case Fig. IV.5**C**, the short range localized connections were removed from **B**, so only uniform random connections were used. As can be seen, the traces are almost exactly the same between regions. Synchrony was computed for N_s uniform chosen neurons from the whole brain as:

$$\chi = \frac{\sigma^2_{V_{\text{avg}}}}{\frac{1}{N_s}\sum_{i=1}^{N_s}\sigma^2_{V_i}},$$

where

$$V_{\text{avg}} = \frac{1}{N_s}\sum_{i=1}^{N_s} V_i$$

is the average voltage of the N_s sampled neurons and $\sigma^2_{V_j}$ measures variance:

$$\sigma^2_{V_j} = \langle V(t)^2\rangle_t - \langle V(t)\rangle_t^2,$$

where $\langle\cdot\rangle_t$ denotes average over t. The synchrony was measure for $N_s = 2^n$ for $n = 6\ldots15$. As can be seen from **D**, although in all cases the whole-cortex synchrony was low, synchrony was an order of magnitude lower in the case of the fully realistic connnectivity. Furthermore, the synchrony was slightly lower when localized connections were added to the uniform random connectivity. In other words, as realism of connections increased, synchrony decreased, suggesting again that realistic connectivity plays an important role in whole-cortex dynamics of the mouse brain.

IV.4: Conclusions

In this work, we combine the most accurate description of the connectivity of all neurons in the mouse cortex to date by merging the CUBIC-Atlas, the connectome of the Allen Brain Atlas, and a large-scale study of the percentage of excitatory and inhibitory neurons in each region. We then develop a simulational framework that can simulate each of these approximately 13×10^6 neurons using a Hodgkin-Huxley type model and their connections

via standard models of synaptic transmission. The behaviors that emerge rely on the details of the connectivity and neuronal ionic, but are much richer than could be predicted based on connectivity or single-cell properties.

One interesting property of our simulated mouse cortex is that it is dynamic. Both the functional connectivity as well as patterns of traveling waves depend on initial conditions or previous stimuli. This property, known to exist *in vivo* based on fMRI, emerges because of the complex interplay between non-trivial single-cell dynamics and a remarkably rich connectivity structure. Detailed connectivity plays a very important role in cortical dynamics (e.g., See Fig. IV.5). Even simplifying the network to 10^4 or 10^5 neurons greatly affects cortical behavior (e.g., Fig. IV.5).

While we show that a framework like ours is needed to understand many cortical behaviors, further work is needed to determine whether our framework itself, which is much more detailed than has previously been used, is itself too simplistic. Our hope is that finer details of single-cell neuronal structure, for example, the transition time of synaptic signals, will not be critical for our goals of studying synchrony across the brain. The effects of paracrine signals and glia also vary but may operate on longer timescales that we study. Likewise, characterizing the heterogeneity of single-cell dynamics between different regions is an important next step. However, it is our hope that focusing on the detailed connections between neurons at scale as well as using the conductance-based single-cell models that have traditionally been used to explain neuronal excitability will capture many phenomena currently measured by EEG or fMRI.

Much of our work focused on creating a platform where this framework could be simulated on desktop computers used by millions of individuals. We hope this will further provide mechanistic explanations for patterns of synchrony across the mouse brain. This is an important step before we pivot to the much more complex dynamics seen in the human brain.

CHAPTER V

Connecting Chapters via a Case Study

V.1: Conclusion

In this part, we attempt to give a high-level summary of the usage and features of our numerical methods and modeling in the form of a case study.

V.1.1: Model Description

The Simplified averaged neuron (SAN) model [YSUT+18, TSS+16] gives a neuronal model where a single neuron can exhibit both a wake-like neuronal firing pattern and a slow-wave-sleep(SWS) neuronal firing pattern depending on the choice of parameters. The SAN model is a particularly interesting example for numerical simulation in that it can display different firing patterns as parameters vary.

The model and parameters here are identical to those in the supplementary information of [YSUT+18]. For reference, they are included here:

We take $\mathbf{x} = (0.01V, n_k, 0.1[Ca^{2+}], Y)$, where V, n_k, $[Ca^{2+}]$ are variables in the SAN model, and Y is a coupling variable that represents both the excitatory and the inhibitory

coupling in the [YSUT+18], as a way to slightly simplify the analysis that follows.

$$C\frac{dV}{dt} = -I_L(V) - I_K(V, n_k) - I_{Ca}(V) - I_{KCa}(V, [Ca^{2+}]) - I_{NaP}(V) - I_c(V, Y) \tag{V.1.1}$$

$$\frac{dn_k}{dt} = 4(\alpha_n(V)(1 - n_k) - \beta_n(V)n_k) \tag{V.1.2}$$

$$\frac{d[Ca^{2+}]}{dt} = -\alpha_{Ca}AI_{Ca}(V) - \frac{[Ca^{2+}]}{\tau_{Ca}} \tag{V.1.3}$$

$$\frac{dY}{dt} = -\frac{Y}{\tau_e} \tag{V.1.4}$$

Where the intermediate functions are defined as

$$I_L(V) = g_L(V - V_L) \tag{V.1.5}$$

$$I_K(V, n_k) = g_k n_k^4 (V - V_K) \tag{V.1.6}$$

$$\alpha_n(V) = 0.01\frac{V + 34}{1 - \exp(-\frac{V+34}{10})} \tag{V.1.7}$$

$$\beta_n(V) = \frac{1}{8}\exp(-\frac{V + 44}{25}) \tag{V.1.8}$$

$$I_{Ca}(V) = g_{Ca}m_{Ca\infty}^2(V)(V - V_{Ca}) \tag{V.1.9}$$

$$m_{Ca\infty}(V) = \frac{1}{1 + \exp(-\frac{V+20}{9})} \tag{V.1.10}$$

$$I_{KCa}(V, [Ca^{2+}]) = g_{KCa}m_{KCa\infty}([Ca^{2+}])(V - V_K) \tag{V.1.11}$$

$$M_{KCa\infty}([Ca^{2+}]) = \frac{1}{1 + (\frac{K_D}{[Ca^{2+}]})^{3.5}} \tag{V.1.12}$$

$$I_{NaP}(V) = g_{NaP}m_{NaP\infty}^3(V)(V - V_{Na}) \tag{V.1.13}$$

$$m_{NaP\infty}(V) = \frac{1}{1 + \exp(-\frac{V+55.7}{7.7})} \tag{V.1.14}$$

$$I_c(V, Y) = g_e Y(V - V_e) + g_i Y\frac{\tau_i}{\tau_e}(V - V_i) \tag{V.1.15}$$

(V.1.16)

and the constants are given as

$C = 1$	$A = 0.02$	$k_D = 30$	
$V_L = -60.95$	$g_L = 0.015915$		
$V_{Na} = 55$	$g_{Na} = 12.2438$	$g_{NaP} = 0.63314$	
$V_K = -100$	$g_K = 19.20436$	$g_{KCa} = 0.7506$	
$V_{Ca} = 120$	$g_{Ca} = 0.1624$	$\tau_{Ca} = 739.09$	$\alpha_{Ca} = 0.5$
$V_e = 0.0$	$g_e = 0.513425$	$\tau_e = 2$	
$V_i = -70$	$g_i = 0.252916$	$\tau_i = 10$	

Table V.1: Constants used in the SAN model.

V.1.2: Applications Using LSKF: Tracking a Single Neuron by Periodic Measurement of Membrane Potential

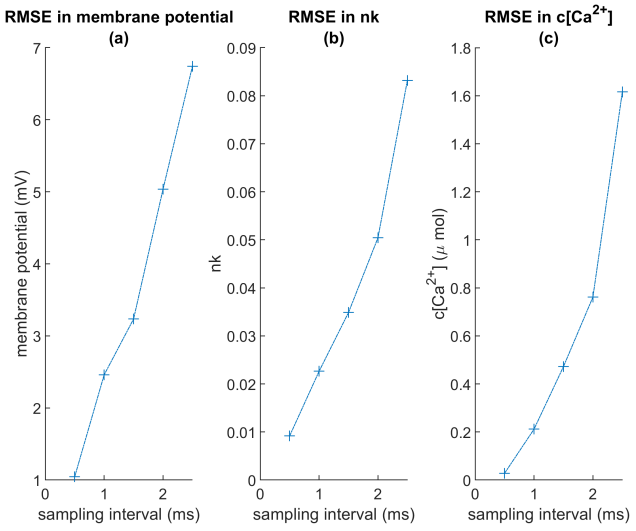


Figure V.1: Estimate status of a single noisy SAN with varying interval between measurements.

Suppose one has a periodic measurement of the membrane potential of a single noisy SAN and wants to infer the other quantities from this measurement. The LSKF can be used for this state estimation problem. In Fig. V.1, it can be noted that even if only membrane potential is measured, the LSKF gives a reasonably good estimation for the other variables:

at a sampling interval of 0.5ms, the RMSE for the nk variable, which ranges between 0 and 1, is less than 0.01 despite the fact that no direct measurement of this variable is obtained. The accuracy of the approximation decreases as the sampling interval increases, as expected.

V.1.3: Applications Using APPD: Macroscopic Behaviors of a Coupled SAN Population

The SAN model is assumed to have a threshold-type coupling with $V_{\text{threshold}} = -30$, with the strength of coupling being a variate over different instances.

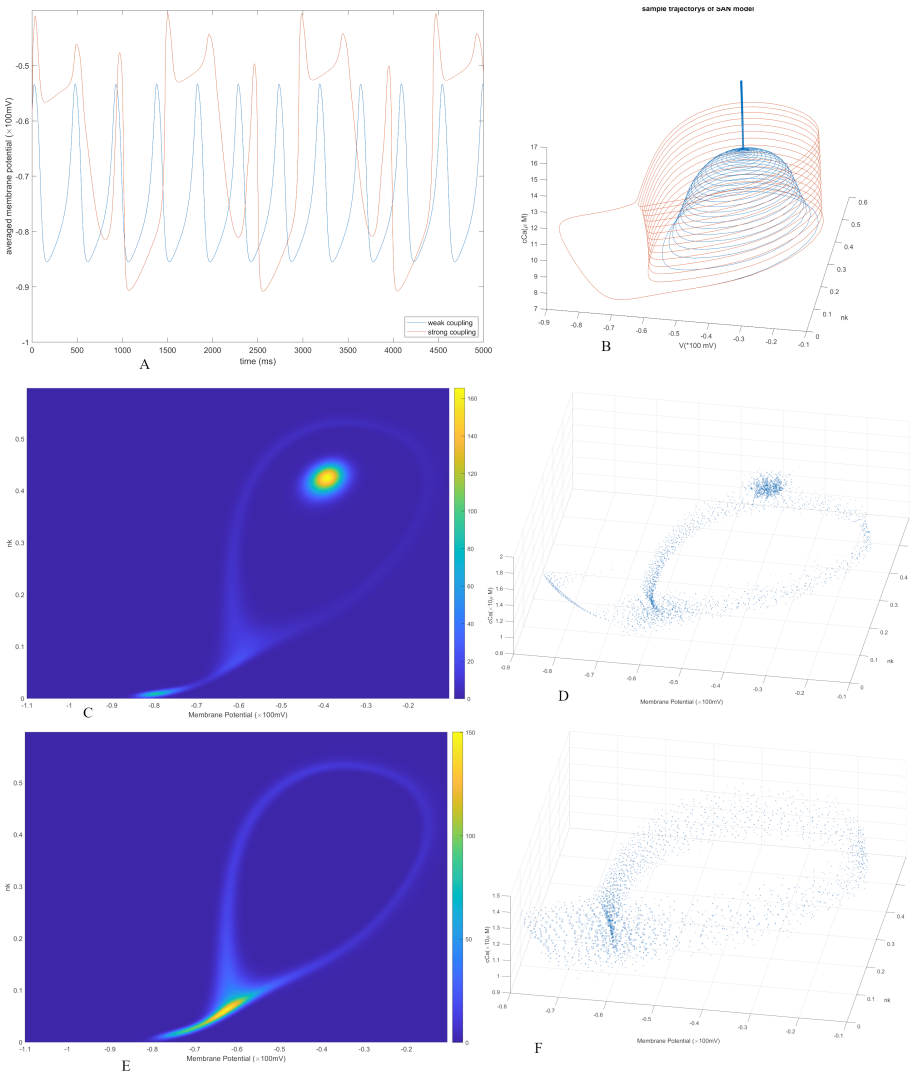


Figure V.2: An example of complex dynamics seen with a cortical model. **A** shows the average membrane potential of the population, for a weak coupling(blue) and strong coupling(orange) instance respectively. **B - F** explains the distinction of behavior. **B** shows the limit cycle trajectory without coupling(orange) and a trajectory under constant coupling input (blue). **C - F** depicts the population density at 500ms under weaker (**E-F**) and stronger (**C-D**) coupling. **C** and **E** show the pdf projected to $V - nk$ plane, whereas **D** and **F** show the distribution of particles in phase space, where the size of the particle represents its weight.

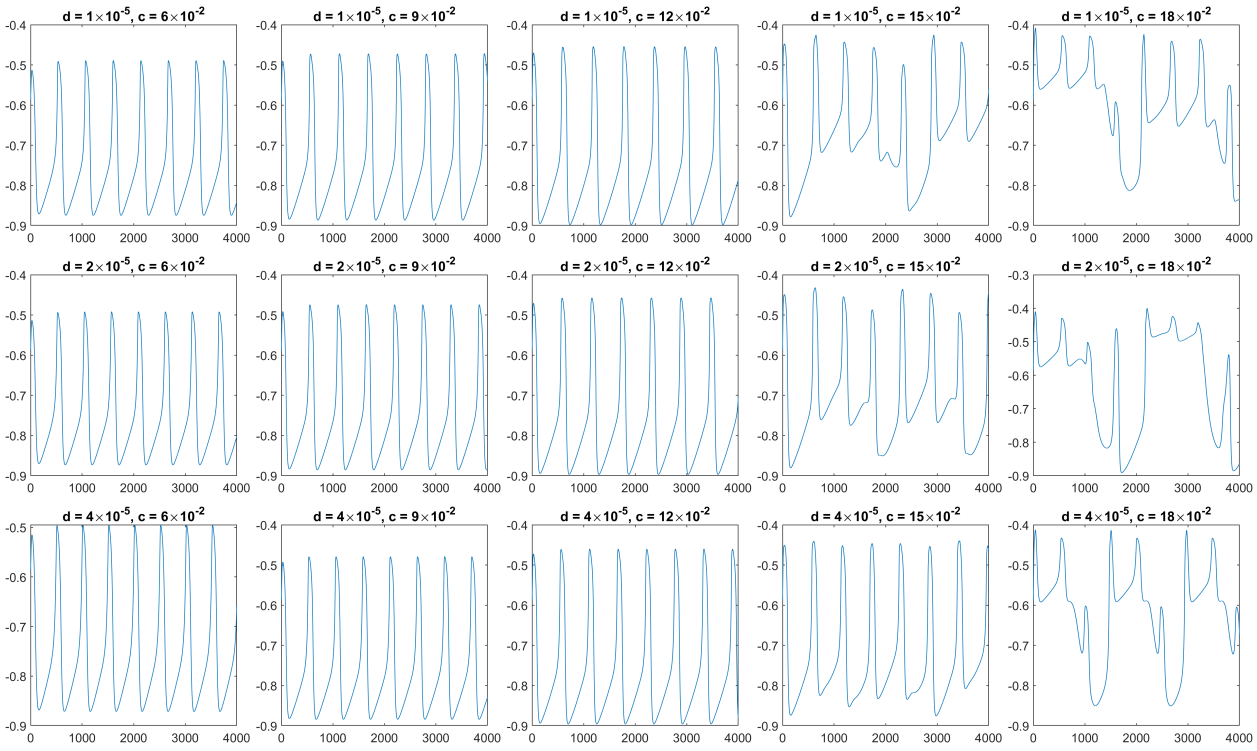


Figure V.3: The averaged membrane potential of SAN populations versus time over a range of parameters.

In the absence of coupling, it has a long limit cycle, with two stages. One stage has frequent firing with increasing intracellular calcium ion concentration and a resting stage with decreasing calcium concentration. However, when the coupling is sufficiently strong, then the neuron will enter a resting state, which is also not stable since coupling would disappear if the entire population is resting. We would like to show that with the APPD, the bifurcation of the model under a population density setting can be reproduced.

From Fig. V.2**A**, it can be observed that the weak coupling case has a shorter and very regular pattern, whereas the strong coupling case has a longer period and more variation between peaks. The different behavior is explained by **B - F** from Fig. V.2. Constant coupling input blocks the firing of neurons, which in turn would reduce coupling input to the population. When the coupling is weak, then neurons follow the behavior closely without

coupling, and the coupling solely synchronizes the neurons. When the coupling is strong, coupling pushes a sub-population to the trajectory under constant coupling input. However, the entire population cannot enter that trajectory since neurons would not be firing in that state, causing the coupling input to disappear. Therefore, it is reasonable to predict that the population would split into groups following the no-coupling trajectory and constant coupling trajectory, which is the result we find. This SAN example deviates greatly from a one-dimensional phase-based population density model; therefore, any method based on a phase-response curve would struggle to accurately represent this system. However, our method can adapt to the bifurcation since it does not assume the population follows a pre-defined limit cycle. Fig. V.3 shows how the coupling coefficient c as well as noise level d affects the population's behavior. As expected, an increased coupling coefficient induces the strong coupling behavior shown in Fig. V.2**C** and **E**. With the coupling coefficient fixed. Moreover, it shows how small noise on the level of individual neurons can lead to large changes in population behavior. Thus, the SAN model shows that the APPD method can capture complicated behaviors.

The above numerical examples used SAN neurons that display a stable SWS pattern. We now turn our focus to the transition between the SWS and wake state for a population of neurons. In [YSUT+18], the Calcium decay rate is the main factor used to control the transition between the wake and the SWS state: as the Calcium decay time τ_{Ca} is decreased, the Calcium concentration in the cell decays faster, and the neuron transitions into the wake state that fires continuously. The transition to the wake state for a single neuron happens when τ_{Ca} is reduced to about 0.5 of the initial value of 739.09. To check the effect of coupling, we consider such populations of wake SAN coupled together.

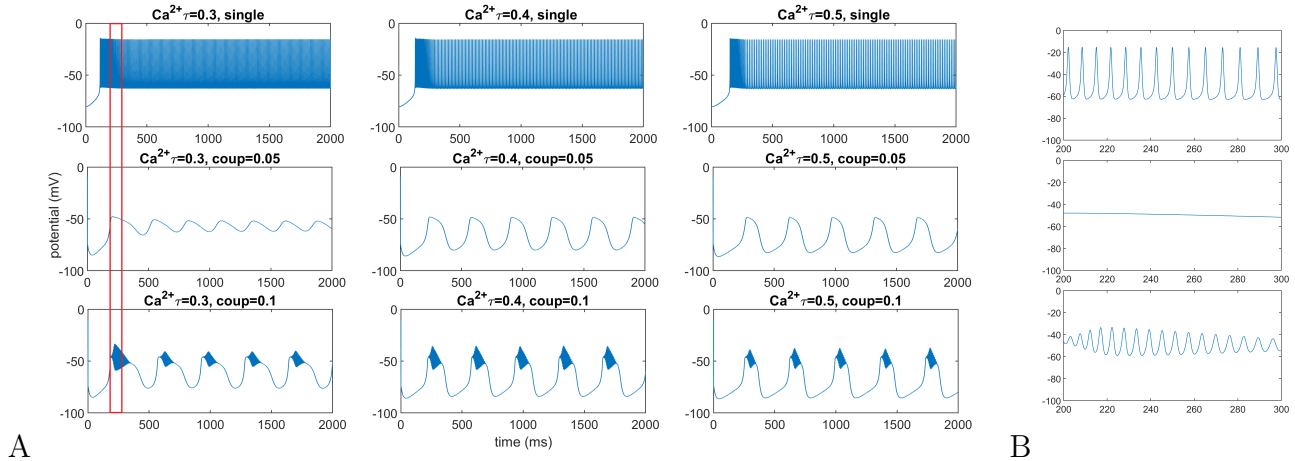


Figure V.4: The average membrane potential of SAN populations as a function of time. In **A**, the first row shows the trajectory of a single SAN, whereas the following rows shows the average membrane potential of a population. From left to right, the Calcium decay time is taken at 0.3, 0.4, and 0.5 of the original value. **B** shows the magnified part in **A** indicated by the red box.

In Fig. V.4, it can be noticed that while the single neuron displays a wake firing pattern, the coupled populations still retains the SWS pattern. Moreover, as shown in panel B of Fig. V.4, a rather weak coupling that does not synchronize the neurons is sufficient to stabilize the SWS pattern. This result indicates that the Calcium decay rate and coupling strength among a population may be sufficient to regulate the sleep-wake transition in a neuronal population.

BIBLIOGRAPHY

[AB02] Elisa Alòs and Stefano Bonaccorsi. Stochastic partial differential equations with dirichlet white-noise boundary conditions. In *Annales de l'IHP Probabilités et statistiques*, volume 38, pages 125–154, 2002.

[ABD14] Remi Abgrall, Héloïse Beaugendre, and Cécile Dobrzynski. An immersed boundary method using unstructured anisotropic mesh adaptation combined with level-sets and penalization techniques. *Journal of Computational Physics*, 257:83–101, 2014.

[AGNR21] Katharine C Abruzzi, Cédric Gobet, Felix Naef, and Michael Rosbash. Comment on "circadian rhythms in the absence of the clock gene bmal1". *Science*, 372(6539), 2021.

[AH09] Ienkaran Arasaratnam and Simon Haykin. Cubature Kalman filters. *IEEE Transactions on automatic control*, 54(6):1254–1269, 2009.

[AHH10] Ienkaran Arasaratnam, Simon Haykin, and Thomas R Hurd. Cubature Kalman filtering for continuous-discrete systems: theory and simulations. *IEEE Transactions on Signal Processing*, 58(10):4977–4993, 2010.

[all] Allen mouse common coordinate framework and reference atlas. Technical report.

[Ara] Ienkaran Arasaratnam. Square-root cubature information filter. Accessed: 2021-01-28.

[BBH21] Adrien Berchet Berchet, Anthony Beaudoin, and Serge Huberson Huberson. Adaptive particle method based on moments for simulating the mass transport in natural flows. *Computational Particle Mechanics*, 8(3):525–534, 2021.

[BH06] Nicolas Brunel and David Hansel. How noise affects the synchronization properties of recurrent networks of inhibitory neurons. *Neural computation*, 18:1066–110, 06 2006.

[BLS95] Agnessa Babloyantz, C Lourenco, and JA Sepulchre. Control of chaos in delay differential equations, in a network of oscillators and in model cortex. *Physica D: Nonlinear Phenomena*, 86(1-2):274–283, 1995.

[BNR15] Irina Bashkirtseva, Alexander B Neiman, and Lev Ryashko. Stochastic sensitivity analysis of noise-induced suppression of firing and giant variability of spiking in a hodgkin-huxley neuron model. *Physical Review E*, 91(5):052920, 2015.

[BS89] Przemyslaw Bogacki and Lawrence F Shampine. A 3 (2) pair of Runge-Kutta formulas. *Applied Mathematics Letters*, 2(4):321–325, 1989.

[BSW77] Jon L. Bentley, Donald F. Stanat, and E.Hollins Williams. The complexity of finding fixed-radius near neighbors. *Information Processing Letters*, 6(6):209 – 212, 1977.

[Buc88] John Buck. Synchronous rhythmic flashing of fireflies. ii. *The Quarterly review of biology*, 63(3):265–289, 1988.

[DF03] Olivier David and Karl J Friston. A neural mass model for meg/eeg:: coupling and neuronal dynamics. *NeuroImage*, 20(3):1743–1755, 2003.

[Don08] Hong Wei Dong. *The Allen reference atlas: A digital color brain atlas of the C57Bl/6J male mouse.* John Wiley & Sons Inc, 2008.

[EGKM18] Csaba Erö, Marc-Oliver Gewaltig, Daniel Keller, and Henry Markram. A cell atlas for the mouse brain. *Frontiers in Neuroinformatics*, 12:84, 2018.

[For17] Daniel B Forger. *Biological clocks, rhythms, and oscillations: the theory of biological timekeeping*. MIT Press, 2017.

[GB08] Dan FM Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics*, 2:5, 2008.

[GH11] Fredrik Gustafsson and Gustaf Hendeby. Some relations between extended and unscented Kalman filters. *IEEE Transactions on Signal Processing*, 60(2):545–555, 2011.

[GSB11] Joshua H Goldwyn and Eric Shea-Brown. The what and where of adding channel noise to the hodgkin-huxley equations. *PLoS Comput Biol*, 7(11):e1002247, 2011.

[HAH+11] Geir Halnes, Sigita Augustinaite, Paul Heggelund, Gaute T Einevoll, and Michele Migliore. A multi-compartment model for interneurons in the dorsal lateral geniculate nucleus. *PLoS Computational Biology*, 7(9):e1002160, 2011.

[HC97] Michael L Hines and Nicholas T Carnevale. The neuron simulation environment. *Neural computation*, 9(6):1179–1209, 1997.

[HHK52] Alan L Hodgkin, Andrew F Huxley, and Bernard Katz. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):424, 1952.

[HSBL18] Bin He, Abbas Sohrabpour, Emery Brown, and Zhongming Liu. Electrophysiological source imaging: a noninvasive window to brain dynamics. *Annual review of biomedical engineering*, 20:171–196, 2018.

[HXL+10] Xiaoying Huang, Weifeng Xu, Jianmin Liang, Kentaroh Takagaki, Xin Gao, and Jian-young Wu. Spiral wave dynamics in neocortex. *Neuron*, 68(5):978–990, 2010.

[Jal03] José Jalife. Rotors and spiral waves in atrial fibrillation. *Journal of cardiovascular electrophysiology*, 14(7):776–780, 2003.

[Jaz07] Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[JBB+10] G Allan Johnson, Alexandra Badea, Jeffrey Brandenburg, Gary Cofer, Boma Fubara, Song Liu, and Jonathan Nissanov. Waxholm space: an image-based reference for coordinating mouse brain research. *Neuroimage*, 53(2):365–372, 2010.

[JR95] Ben H Jansen and Vincent G Rit. Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns. *Biological cybernetics*, 73(4):357–366, 1995.

[JU97] Simon J Julier and Jeffrey K Uhlmann. New extension of the Kalman filter to nonlinear systems. In Ivan Kadar, editor, *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pages 182 – 193. International Society for Optics and Photonics, SPIE, 1997.

[JU04] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[KB61] Rudolph E Kalman and Richard S Bucy. New results in linear filtering and prediction theory. 1961.

[KDPN06] Natalia Krepysheva, Liliana Di Pietro, and Marie-Christine Néel. Space-

fractional advection-diffusion and reflective boundary condition. *Physical Review E*, 73(2):021104, 2006.

[KF12] Jae Kyoung Kim and Daniel B Forger. A mechanism for robust circadian timekeeping via stoichiometric balance. *Molecular systems biology*, 8(1):630, 2012.

[KHG+19] Joseph E. Knox, Kameron Decker Harris, Nile Graddis, Jennifer D. Whitesell, Hongkui Zeng, Julie A. Harris, Eric Shea-Brown, and Stefan Mihalas. High-resolution data-driven model of the mouse connectome. *Network Neuroscience*, 3(1):217–236, 2019.

[KK17] G Yu Kulikov and Maria V Kulikova. Accurate continuous–discrete unscented Kalman filtering for estimation of nonlinear continuous-time stochastic models in radar tracking. *Signal Processing*, 139:25–35, 2017.

[Kur03] Yoshiki Kuramoto. *Chemical oscillations, waves, and turbulence*. Courier Corporation, 2003.

[KYP+17] Yongsoo Kim, Guangyu Robert Yang, Kith Pradhan, Kannan Umadevi Venkataraju, Mihail Bota, Luis Carlos García Del Molino, Greg Fitzgerald, Keerthi Ram, Miao He, Jesse Maurica Levine, et al. Brain-wide maps reveal stereotyped cell-type-based cortical architecture and subcortical sexual dimorphism. *Cell*, 171(2):456–469, 2017.

[KYW+10] Caroline H Ko, Yujiro R Yamada, David K Welsh, Ethan D Buhr, Andrew C Liu, Eric E Zhang, Martin R Ralph, Steve A Kay, Daniel B Forger, and Joseph S Takahashi. Emergence of noise-induced oscillations in the central circadian pacemaker. *PLoS Biol*, 8(10):e1000513, 2010.

[LHA+07] Ed S Lein, Michael J Hawrylycz, Nancy Ao, Mikael Ayres, Amy Bensinger, Amy Bernard, Andrew F Boe, Mark S Boguski, Kevin S Brockway, Emi J Byrnes,

et al. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124):168–176, 2007.

[LSRJ11] Magnus Linderoth, Kristian Soltesz, Anders Robertsson, and Rolf Johansson. Initialization of the Kalman filter without assumptions on the initial state. In *2011 IEEE International Conference on Robotics and Automation*, pages 4992–4997. IEEE, 2011.

[Mar06] Henry Markram. The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[MG02] Sylvie Mas-Gallic. The diffusion velocity method: a deterministic way of moving the nodes for solving diffusion equations. *Transport Theory and Statistical Physics*, 31(4-6):595–605, 2002.

[MMK+21] Tomoyuki Mano, Ken Murata, Kazuhiro Kon, Chika Shimizu, Hiroaki Ono, Shoi Shi, Rikuhiro G. Yamada, Kazunari Miyamichi, Etsuo A. Susaki, Kazushige Touhara, and Hiroki R. Ueda. Cubic-cloud provides an integrative computational framework toward community-driven whole-mouse-brain mapping. *Cell Reports Methods*, 1(2):100038, 2021.

[MMS+18] Tatsuya C Murakami, Tomoyuki Mano, Shu Saikawa, Shuhei A Horiguchi, Daichi Shigeta, Kousuke Baba, Hiroshi Sekiya, Yoshihiro Shimizu, Kenji F Tanaka, Hiroshi Kiyonari, et al. A three-dimensional single-cell-resolution whole-brain atlas using cubic-x expansion microscopy and tissue clearing. *Nature neuroscience*, 21(4):625–637, 2018.

[MSS+19] Christine M Muheim, Andrea Spinnler, Tina Sartorius, Roland Dürr, Reto Huber, Clement Kabagema, Peter Ruth, and Steven A Brown. Dynamic-and frequency-specific regulation of sleep oscillations by cortical potassium channels. *Current Biology*, 29(18):2983–2992, 2019.

[MWJB17] Francesca Melozzi, Marmaduke M Woodman, Viktor K Jirsa, and Christophe Bernard. The virtual mouse brain: A computational neuroinformatics platform to study whole mouse brain dynamics. *Eneuro*, 4(3), 2017.

[NCAB21] Elan Ness-Cohn, Ravi Allada, and Rosemary Braun. Comment on "circadian rhythms in the absence of the clock gene bmal1". *Science*, 372(6539), 2021.

[New91] Nigel J. Newton. Asymptotically efficient Runge-Kutta methods for a class of ito and stratonovich equations. *SIAM Journal on Applied Mathematics*, 51(2):542–567, 1991.

[NK10] Ken H Nagai and Hiroshi Kori. Noise-induced synchronization of a large population of globally coupled nonidentical oscillators. *Physical Review E*, 81(6):065202, 2010.

[NSS+16] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.

[NT00] Duane Q Nykamp and Daniel Tranchina. A population density approach that facilitates large-scale modeling of neural networks: Analysis and an application to orientation tuning. *Journal of computational neuroscience*, 8(1):19–50, 2000.

[OA08] Edward Ott and Thomas M Antonsen. Low dimensional behavior of large systems of globally coupled oscillators. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):037113, 2008.

[OHN+14] Seung Wook Oh, Julie A Harris, Lydia Ng, Brent Winslow, Nicholas Cain, Stefan Mihalas, Quanxin Wang, Chris Lau, Leonard Kuan, Alex M Henry,

et al. A mesoscale connectome of the mouse brain. *Nature*, 508(7495):207, 2014.

[PM14] Diego Pazó and Ernest Montbrió. Low-dimensional dynamics of populations of pulse-coupled oscillators. *Physical Review X*, 4(1):011009, 2014.

[Ros96] Louis F Rossi. Resurrecting core spreading vortex methods: a new scheme that is both deterministic and convergent. *SIAM Journal on Scientific Computing*, 17(2):370–397, 1996.

[Ros05] Louis F Rossi. Achieving high-order convergence rates with deforming basis functions. *SIAM Journal on Scientific Computing*, 26(3):885–906, 2005.

[Row07] Peter Rowat. Interspike interval statistics in the stochastic hodgkin-huxley model: Coexistence of gamma frequency bursts and highly irregular firing. *Neural Computation*, 19(5):1215–1250, 2007.

[RVS+20] Sandipan Ray, Utham K Valekunja, Alessandra Stangherlin, Steven A Howell, Ambrosius P Snijders, Gopinath Damodaran, and Akhilesh B Reddy. Circadian rhythms in the absence of the clock gene bmal1. *Science*, 367(6479):800–806, 2020.

[San96] Terence David Sanger. Probability density estimation for the interpretation of neural population codes. *Journal of neurophysiology*, 76(4):2790–2793, 1996.

[Sar07] Simo Sarkka. On unscented Kalman filtering for state estimation of continuous-time nonlinear systems. *IEEE Transactions on automatic control*, 52(9):1631–1641, 2007.

[Sär13] Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

[Set85] James A Sethian. Curvature and the evolution of fronts. *Communications in Mathematical Physics*, 101(4):487–499, 1985.

[SF16] Adam R. Stinchcombe and Daniel B. Forger. An efficient method for simulation of noisy coupled multi-dimensional oscillators. *Journal of Computational Physics*, 321:932 – 946, 2016.

[SLKW+13] Paula Sanz Leon, Stuart A Knock, M Marmaduke Woodman, Lia Domide, Jochen Mersmann, Anthony R McIntosh, and Viktor Jirsa. The virtual brain: a simulator of primate brain network dynamics. *Frontiers in neuroinformatics*, 7:10, 2013.

[SS12] Simo Särkkä and Arno Solin. On continuous-discrete cubature Kalman filtering. *IFAC Proceedings Volumes*, 45(16):1221–1226, 2012.

[THHDI07] Tsz-Leung To, Michael A Henson, Erik D Herzog, and Francis J Doyle III. A molecular model for intercellular synchronization in the mammalian circadian clock. *Biophysical journal*, 92(11):3792–3803, 2007.

[TJ00] Paul H. E. Tiesinga and Jorge V José. Synchronous clusters in a noisy inhibitory neural network. *Journal of Computational Neuroscience*, 9(1):49–65, 2000.

[TSS+16] Fumiya Tatsuki, Genshiro A Sunagawa, Shoi Shi, Etsuo A Susaki, Hiroko Yukinaga, Dimitri Perrin, Kenta Sumiyama, Maki Ukai-Tadenuma, Hiroshi Fujishima, Rei-ichiro Ohno, et al. Involvement of ca2+-dependent hyperpolarization in sleep duration in mammals. *Neuron*, 90(1):70–85, 2016.

[WCW00] Yuqing Wang, David TW Chik, and ZD Wang. Coherence resonance and noise-induced synchronization in globally coupled hodgkin-huxley neurons. *Physical Review E*, 61(1):740, 2000.

[WF21a] Ningyuan Wang and Daniel B Forger. The asymmetric particle population density method for simulation of coupled noisy oscillators, 2021.

[WF21b] Ningyuan Wang and Daniel B Forger. The level set Kalman filter for state estimation of continuous-discrete systems. *arXiv preprint arXiv:2103.11130*, 2021.

[Win67] Arthur T Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of theoretical biology*, 16(1):15–42, 1967.

[XPP+14] Zhihua Xie, Dimitrios Pavlidis, James R Percival, Jefferson LMA Gomes, Christopher C Pain, and Omar K Matar. Adaptive unstructured mesh modelling of multiphase flows. *International journal of multiphase flow*, 67:104–110, 2014.

[YB16] Rolf JF Ypma and Edward T Bullmore. Statistical analysis of tract-tracing experiments demonstrates a dense, complex cortical network in the mouse. *PLoS computational biology*, 12(9):e1005104, 2016.

[YDW+19] Shuangming Yang, Bin Deng, Jiang Wang, Huiyan Li, Meili Lu, Yanqiu Che, Xile Wei, and Kenneth A Loparo. Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons. *IEEE transactions on neural networks and learning systems*, 31(1):148–162, 2019.

[YSUT+18] Kensuke Yoshida, Shoi Shi, Maki Ukai-Tadenuma, Hiroshi Fujishima, Rei-ichiro Ohno, and Hiroki R Ueda. Leak potassium channels regulate sleep duration. *Proceedings of the National Academy of Sciences*, 115(40):E9459–E9468, 2018.

[YTN16] Esin Yavuz, James Turner, and Thomas Nowotny. Genn: a code generation framework for accelerated brain simulations. *Scientific reports*, 6(1):1–14, 2016.